Unit 1

Introduction of Database

- Purpose of database system
- View of data
- Database languages
- Transaction management
- Database administrator
- System structure

Data:

Data is commonly defined as raw facts or observation, typically about physical phenomena or business transactions. For example of data would be the marks obtained by students in different subjects. Data can be in any form-numerical, textual, graphical, image, sound, video etc.

The following figure shows the hierarchy of data storage.

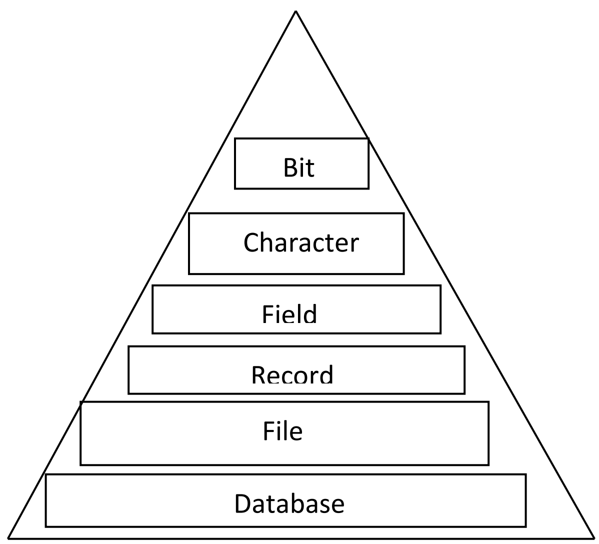


Fig:- Data storage hierarchy

Information:

Information is defined as refined or processed data that has been transformed into meaningful and useful form for specific users. For example, after processing the marks obtained by student it transformed into information, which is meaningful and from which we can decide which student stood first, second and so forth. Information comes from data and takes the form of table, graphs, diagrams etc.

Database:

A database is an organized collection of data and contains information relevant to an enterprise. The database is also called the repository or container for a collection of data files. For example, **university database** for maintaining information about *students*, *courses* and *grades* in university.

Characteristics of data in database:

The data in a database should have the following features:

- **Shared:** Data should be sharable among different users and applications.
- **Persistence:** Data should exist permanently in the database. Changes in the database must not be lost because of any failure.
- **Validity/Integrity/Correctness:** It should maintain the integrity so that there is always correct data in the database.
- **Security:** Data should be protected from unauthorized access.
- **Non-redundancy:** Data should not be repeated.
- **Consistency:** A consistent state of the database satisfies all the constraints specified in the database. Data in a database is consistent if any changes in the database take the database from one consistent state to another.
- **Independence:** The three levels in the schema should be independent of each other so that the changes in the schema in one level should not affect the other levels.

Database Management System:

A Database Management System (DBMS) is a collection of interrelated data and the set of programs to access those data. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

For example, in a computer system, the checking account processing system, the auto loan system and the saving accounts would have a common database. This database based approach to data processing is shown in fig below:

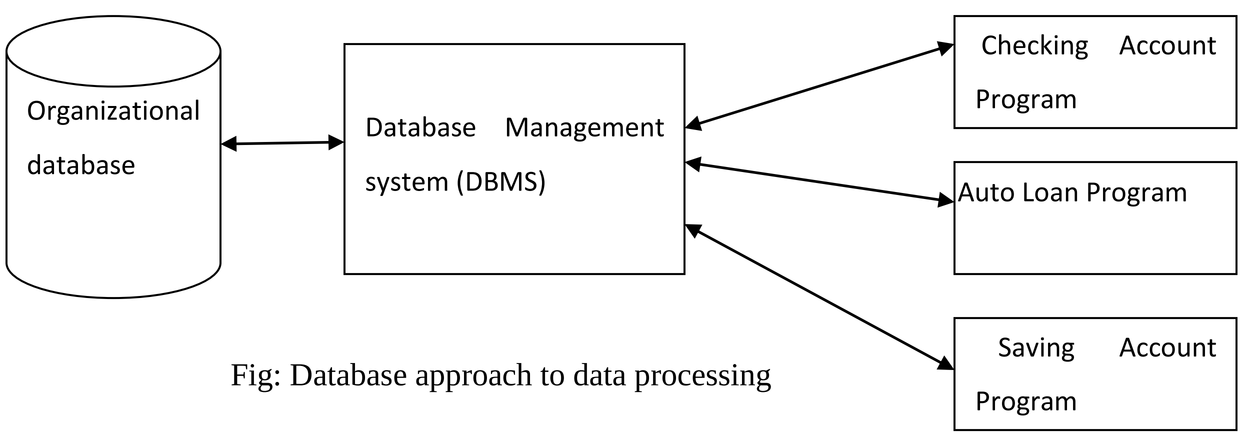


Fig: Database approach to data processing

Objective of DBMS

The DBMS is able

- To provide large space or storage for relevant data.
- To provide easy access to the data for the users.
- To provide quick response to user request for any data.
- To remove duplicate (redundant) data.
- To update the database latest modification immediately.
- To allow the multiple users to be active at one time.
- As the organization grows, DBMS allows the growth of the database system.
- To provide maximum protection to data from any physical damage and unauthorized access.

Database system:

A ***database system*** consists of database, database Management system, and application program. A database system is just a computerized record keeping system. Database is a repository for a collection of computerized data files. Users of database system can perform a variety of operation on such file.

Some common examples of the DBMS software are Oracle, Sybase, Microsoft SQL Server, DB2, MySQL, Postgres, Dbase, Ms-Access etc.

Applications of DBMS:

Databases form an essential part of almost all enterprises. Some database applications are given below:

- **Banking:** For customer information and all transactions
- **Airlines:** For reservations and schedules information
- **Universities:** For the student information , course registration and grades
- **Credit and transaction:** For purchase credit cards and generates monthly statement
- **Telecommunication:** Keeping records of all the telephone calls, generating monthly bills etc.
- **Finance:** For storing financial information
- **Sales :**For customers, products and purchases information
- **Manufacturing:** For tacking production, inventory, orders, supply chain management
- **Human resources:** For storing the information about employee records, salaries, tax deductions

Flat-file Systems:

A flat file system stores data in a plain text file. A flat file is a file that contains records, and in which each record is specified in a single line. Fields from each record may simply have a fixed width with padding, or may be delimited by whitespace, tabs, commas or other characters. Extra formatting may be needed to avoid delimiter collision. There are no structural

relationships. The data are "flat" as in a sheet of paper, in contrast to more complex models such as a relational database.

For example, in a computer system, the checking account processing system would have its own data files. This file based approach to data processing is shown in fig below:

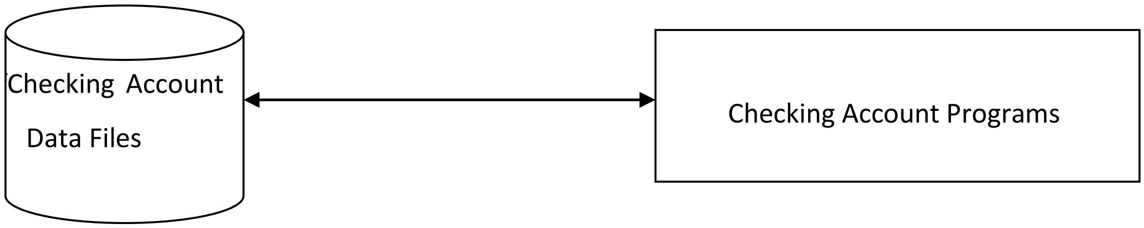


Fig: File-based approach to data processing

Limitations of Flat File System:

Keeping organizational information in a file-processing system has a number of major disadvantages:

- **Data redundancy**: The address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost.
- **Data inconsistency**: The various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.
- **Difficulty in accessing data:** Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
- **Data isolation**: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
- **Integrity problems**: The problem of integrity is the problem of ensuring that the data in database is correct after and before the transaction. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). When new constraints are added, we have to change the programs to enforce them.
- **Atomicity problems:** Execution of transactions must be atomic. This means transactions must execute at its entirety or not at all. Consider a program to transfer $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *B*, resulting in an inconsistent database state. It is difficult to ensure atomicity in a conventional file-processing system.
- **Concurrent-access anomalies**: Concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state, if the programs executing on behalf of each withdrawal read the old balance
- **Security problems**: Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do

not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult in flat file system.

Purpose of DBMS (Functions of DBMS):

The benefits of using DBMS are:

- **To reduce redundancy:** Repeating of the same information in database is called redundancy of data which leads to several problems such as wastage of space, duplication effort for entering data and inconsistency. When DBMS is used and database is created, redundancy is minimized.
- **To avoid inconsistency:** The database is said to be inconsistent if various copies of the same data may no longer agree. For example, a changed customer address may be reflected in saving account but not elsewhere in the system. By using DBMS we can avoid inconsistency.
- **To share data:** The data in the database can be shared among many users and applications. The data requirements of new applications may be satisfied without having to create any new stored files.
- **To provide support for transactions:** A transaction is a sequence of database operations that represents a logical unit of work. It accesses a database and transforms it from one state to another. A transaction can update a record, delete one, modify a set of records etc. when the DBMS does a 'commit', the changes made by the transaction are made permanent. We can roll back the transaction to undo the effects of transaction.
- **To maintain integrity:** Most database applications have certain integrity constraints that must hold for the data. A DBMS provides capabilities for defining and enforcing these constraints. For example, the value of roll number field of each student in student database should be unique for each student. It is a type of rule. Such a rule is enforced using constraint at the time of creation of database.
- **To enforce security:** Not every user of the database system should be able to access all data. Different checks can be established for each type of access (retrieve, modify, delete, etc) to each piece of information in the database.
- **To provide efficient backup and recovery:** Provide facilities for recovering from software and hardware failures to restore database to previous consistent state.
- **To Concurrent Access Database:** Concurrent access means access to the same data simultaneously by more than one user. The same data may be used by many users for the purpose reading at the same time. But when a user tries to modify a data, there should be a concurrency control mechanism to avoid the inconsistency of data. A DBMS provides facilities for these operations.

Disadvantages of DBMS

- **Problem associated with centralization:** Centralization increases the security problems.
- **Cost of software:** Today's there are several softwares which are very costly. Hence from economic point of view it is the drawback.

- **Cost of hardware:** To support various software some upgraded hardware components are needed. Hence from economic point of view it is the drawback.
- **Complexity of backup and recovery:** DBMS provides the centralization of the data, which requires the adequate backups of data.
- Overhead for providing generality, security, recovery, integrity, and concurrency control.
- If the database and applications are simple, well defined, and not expected to change.
- If there are stringent real-time requirements that may not be met because of DBMS overhead.

Differences bet^{n} DBMS and file processing system:

DBMS	File processing system
1. A Database Management System (DBMS) is a collection of interrelated data and the set of programs to access those data.	1. A flat file system stores data in a plain text file. A flat file is a file that contains records, and in which each record is specified in a single line.
2. Data redundancy problem is not found.	2. Data redundancy problem exist.
3. Data inconsistency does not exist.	3. Data inconsistency may exist.
4. Accessing data from database is easier.	4. Accessing data from database is comparatively difficult.
5. The problem of data isolations is not found.	5. Here data are scattered in various files and formats so data isolation problem exist.
6. Atomicity and integrating problems are not found.	6. Here these problems are found.
7. Data are more secure.	7. Data are less secure.
8. Concurrent access and crash recovery.	8. Here there is no concurrent access and no recovery.

Views of Data/ Data abstraction:

The system hides certain details of how the data are stored and maintained and such view is an abstract view.

✓ The Database System provides users with an *abstract view* of the data.

Data Abstraction:- ***The database designers use the complex data structure to represent the data in the database and developer hides the complexity from user from several level of abstraction such as physical level, logical level, and view level. This process is called data abstraction***.

Levels of Data Abstraction:
The three levels of data abstraction can be shown as follows:

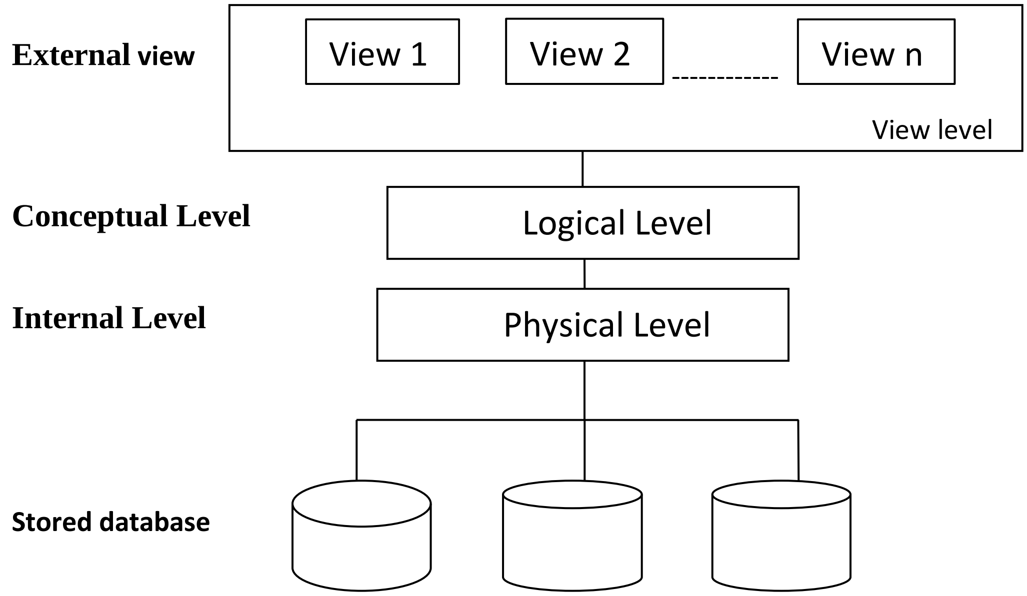


Fig: Three levels of data abstraction

Physical level

- ✓ It is the lowest level of abstractions describes how the data are actually stored.
- ✓ The physical level describes complex low level data structure in details.
- ✓ At this level records such as customer, account can be described as a block of consecutive storage location (e.g. byte, word)
- ✓ The database system hides many of the lowest level storage details from database programmer. Database administrator may be aware of certain details of the physical organization of the data.

Logical level

- ✓ It is the next higher level of data abstraction which describes what data are stored in the database, and what relationships exist among those data.
- ✓ At the logical level , each record is described by a type definition
- ✓ Programmers and database administrator works at this level of abstraction.

View level

- ✓ It is the highest level of abstraction describes only a part of the database and hides some information to the user.
- ✓ At view level, computer users see a set of application programs that hide details of data types. Similarly at the view level several views of the database are defined and database user see only these views.
- ✓ Views also provides the security mechanism to prevent users from accessing certain parts of the database (that is views can also hide information (such as an employee‘s salary) for security purposes.)

Instances and Schemas

Instance (Database State):

The collection of information stored in the database at a particular moment is called an instance of the database. It is the actual content of the database at a particular point in time

✓ The term *instance* is also applied to individual database components, e.g. *record instance, table instance, entity instance.*

Initial Database State

Refers to the database state when it is initially loaded into the system.

Valid State

A state that satisfies the structure and constraints of the database.

Schema:

The overall design of the database which is not expected to change frequently is called database schema. Simply, the database schema is the logical structure of the database.

- ✓ The concept of database schema and instances can be understood by analogy to a program written in a programming language
- ✓ A database schema corresponds to the variable declaration and the values of the variables in a program at a point in time correspond to an instance of a database.
- ✓ The database systems have several schemas and partitioned according to the level of abstraction such as physical and logical schema.

STUDENT

Name	Student-number	Class	Major

Fig: Schema diagram for Student

> Note:
> - ✓ The ***database schema*** changes very infrequently.
> - ✓ The ***database state*** changes every time the database is updated.

Data Independence:

The three schema architecture further explains the concept of data independence, the capacity to change the schema at one level without having to change the schema at the next higher level.

- ➢ **Logical Data Independence**
- ➢ **Physical Data Independence**

Logical Data Independence:

The capacity to change the conceptual schema without having to change the external schemas and their associated application programs is called logical data independence. When modification is done to the conceptual schema (i.e tables) the mapping called "external mapping" is changes automatically by DBMS.

Physical Data Independence:

The capacity to change the internal schema without having to change the conceptual schema is called physical data independence. When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS. This mapping is called "logical mapping".

For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance.

Database Languages:

DBMS provides two languages: Data-Definition Language (DDL) and Data-Manipulation Language (DML).

Data Definition Language (DDL):

Data definition language is the specification notation for defining the database schema.

- ✓ Used by the DBA and database designers to specify the conceptual schema of a database.
- ✓ In many DBMSs, the DDL is also used to define internal and external schemas (views).

Example:

```
CREATE TABLE account
 (
        account-number      CHAR(10),
        balance             INTEGER
 )
```

- ✓ Execution of the above DDL statement creates the account table.
- ✓ It updates a special set of tables called the data dictionary.

Data dictionary: DDL compiler generates a set of tables stored in a *data dictionary*. *Simply*, **Data dictionary *is a special set of tables that contain the information about tables.*** Data dictionary contains metadata (i.e., data about data)

- **Metadata**: ***data that describes the database or one of its parts is called metadata. The schema of a table is an example of metadata***

The **DDL provides** the facilities to define:

- ✓ **Database scheme**
- ✓ **Database tables**
- ✓ **Integrity constraints**
    - Domain constraints
    - Referential integrity (references constraint in SQL)
    - Assertions
    - Triggers
    - Views
- ✓ **Security and Authorization**
- ✓ **Modify the Scheme**

The common DDL Commands are: CREATE, ALTER, DROP

Data Manipulation Language (DML):

A **Data-manipulation language** (DML***) is a language that enables users to access or manipulate data organized by the appropriate data model***.DML also known as query language.

Ex. SELECT *
FROM account
WHERE balance <1000

Execution of this statement retrieves the records of all accounts in which balance is below 1000.
There are basically two classes of DML:

Procedural DMLs (or Low-level DML):
In procedural DMLs, a user specifies what data are required and how to get those data.

Declarative (or nonprocedural or high-level) DMLs:
In declarative DMLs a user specifies what data are needed without specifying how to get those data

The data manipulation is:

- ✓ The retrieval of information stored in the database
- ✓ The insertion of new information into the database
- ✓ The deletion of information from the database
- ✓ The modification of information stored in the database

The common DML commands: SELECT, INSERT, UPDATE, DELETE

Query: ***A query is a statement requesting the retrieval of information***. SQL is the most widely used query language Select, insert, update, delete etc are the SQL DML statement

Data Manipulation Language:

By Data Manipulation, we mean

- ➢ The retrieval of information stored in the database.
- ➢ The insertion of new information into the database.
- ➢ The deletion of information from the database.
- ➢ The modification of information stored in the database.

There are basically two types of DML:

- ✓ **Procedural DMLs**: require the user to specify what data are needed and how to get those data.
- ✓ **Non-Procedural DMLs**: require a user to specify what data are needed without specifying how to get those data.

Transaction Management:

Collections of operations that form a single logical unit of work are called transactions. For example, a transfer of funds from a checking account to a savings account is a single operation from the customer’s standpoint; within the database system, however, it consists of several operations. A database system must ensure proper execution of transactions despite failures—either the entire transaction executes, or none of it does.
Consider the transaction,

A = A – 50;
Write (A);
Read (B);
B = B+50;
Write (B);

To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

- ✓ **Atomicity:** Either all operations of the transaction are reflected properly in the database or none are. Suppose during the execution of above transaction a failure occurred after the write (A) operation but before write (B) operation. Then the values of amount reflected in database will be 950 and 2000. The system destroyed 50 as a result of failure.

- ✓ **Consistency:** Database must be in correct state before and after execution of the transaction. The consistency requirement here is that sum of A and B be unchanged by the execution of transaction. Without the consistency requirement, money could be created or destroyed by a transaction.

- ✓ **Isolation:** Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

- ✓ **Durability:** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

These properties are called ACID properties, with acronym derived from the first letters of the above four properties.

Transactions access data using two operations:

- ✓ **Read(x):** Which transfers the data item x from the database to a local buffer belonging to the transaction that executed the read operation.
- ✓ **Write(x):** Which transfers the data item x from the local buffer of the transaction that executed to the database.

Database users and administrators

Database users:

Users are differentiated by the way they expect to interact with the system. There are four different types of database-system users:

- ✓ **Naive Users**: They are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example: a bank teller who needs to transfer $50 from account A to account B invokes a program called transfer.

- ✓ **Application programmers:** They are computer professionals who write application programs. Application programmers can choose any tools to develop user interface.
- ✓ **Sophisticated users:** They interact with the system without writing programs. They form their requests in a database query language.
- ✓ **Specialized users**: They are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (graphics data and audio data), and environment-modeling systems.

Database Administrator:

The person who has such central control over the system is called the database administrator (DBA). The function of the DBA includes the following:

- ✓ **Schema definition**: The DBA creates the original database schema by writing a set of definitions that is translated by the DDL compiler to a set of tables that is stored permanently in the data dictionary.
- ✓ **Schema and physical-organization modification**: The DBA carries out changes to the schema and physical organization to reflect the changing needs to the organization, or to alter the physical organization to improve performance.
- ✓ **Granting the authorization of data access**: The granting of different types of privileges to the database users so that all the users are not able to all data.
- ✓ **Integrity-constraint specifications**: The data values stored in the database must satisfy certain consistency constraints. The database administrator must specify such a constraint explicitly.
- ✓ **Routine maintenance:** Routine maintenance includes periodic backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding, Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required etc.

Database Models:

Data model is a collection of tools for describing data, data relationships, data semantics and data constraints. The database model refers the way for organizing and structuring the data in the database. Traditionally, there are different database models which are used to design and develop the database of the organization.

1. **Entity- Relationship Model**
2. **Object oriented Model**

} **Object based data model**

3. **Relational Model**
4. **Hierarchical model**
5. **Network Model**
6. **Object Relational Data Model**

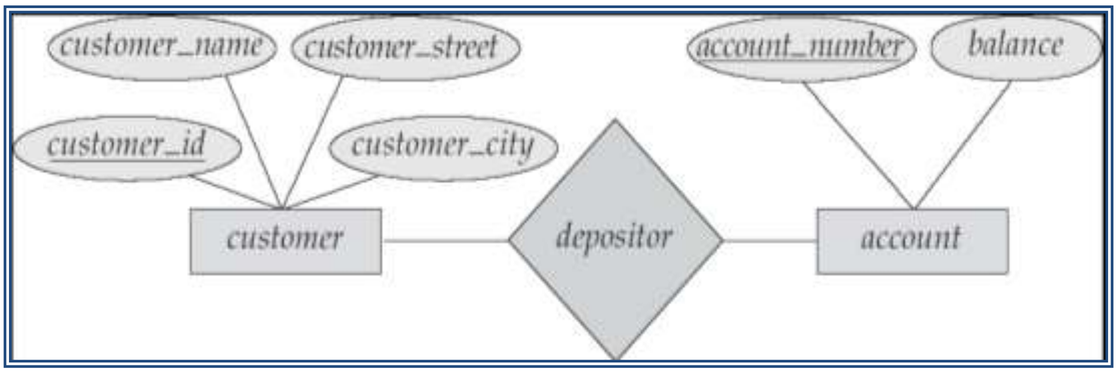
} **Record based data models**

1. Entity- Relationship Model:

The E-R data models is based on a perception of real world that consist of a collection of basic objects called entities and relationship among these objects. In an E-R model a **database** can be modeled as a collection of ***entities, and relationship*** among entities.

Overall logical structure of a database can be expressed graphically by E-R diagram. The basic components of this diagram are:

- **Rectangles** (represent entity sets)
- **Ellipses** (represent attributes)
- **Diamonds** (represent relationship sets among entity sets)
- **Lines** (link attributes to entity sets and entity sets to relationship sets)



2. Relational Model:

It is the current favorite model. The relational model is a lower level model that uses a collection of tables to represent both data and relationships among those data. Each table has multiple columns, and each column has a unique name. Each table corresponds to an entity set or relationship set, and each row represents an instance of that entity set or relationship set. Structured query language (SQL) is used to manipulate data stored in tables.

Customer

Customer_id	Customer_name	Customer_street	Customer_city
1	Bhupi	Chandani	Kanchanpur
2	Arjun	Balkhu	Kathmandu
3	Abin	Pulchoak	Lalitpur

Depositor (Relationship table)

Customer_id	Account_no
1	Ac-33
2	Ac-12
3	Ac-65
3	Ac-77
2	Ac-33

Account

Account_no	Balance
Ac-33	10000
Ac-65	20000
Ac-12	70000
Ac-77	9000

3. Object oriented data model:

The object oriented data model is based on object-oriented programming paradigm. It is based on the concept of encapsulating the data and the functions that operate on those data in a single unit called an object. The internal parts of objects are not visible externally.
Here one object communicates with other objects by sending message.

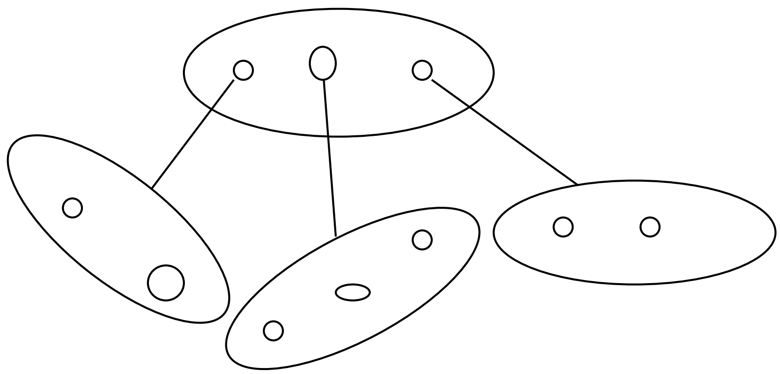


Fig: Object oriented data model

4. Hierarchical data model:

In a hierarchical data model, the data elements are linked in the form of an inverted tree structure with the root at the top and the branches formed below. Below the single root data element are subordinate elements, each of which, in turn, has one or more other elements. There is a parent child relationship among the data elements of a hierarchical database. There may be many child elements under each parent element, but there can be only one parent element for any child element. The branches in the tree are not connected.

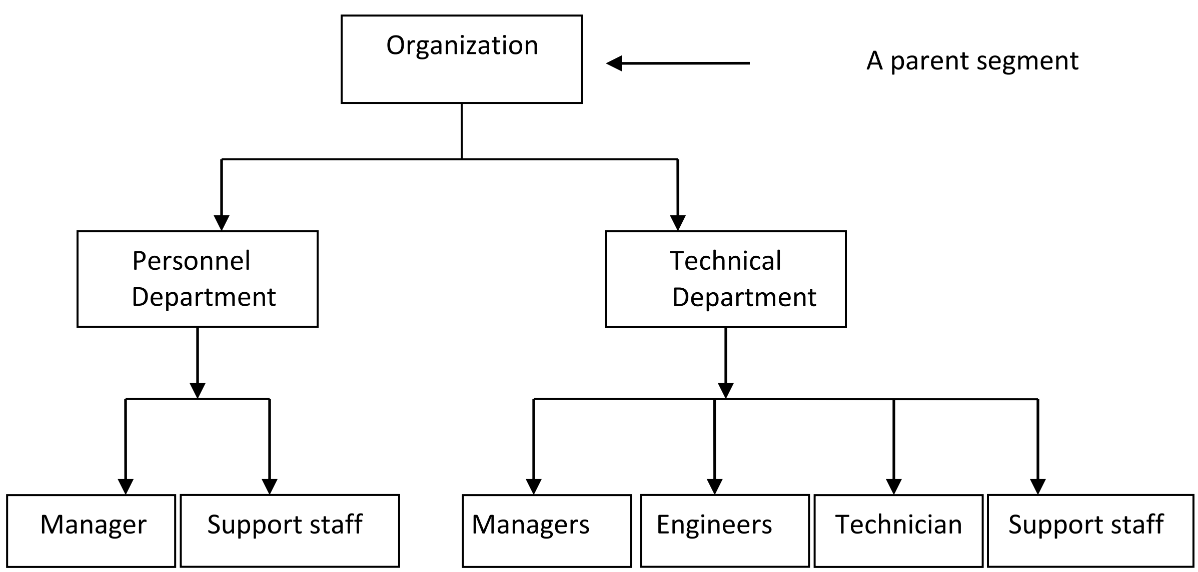


Fig: An example of hierarchical database model

The main limitation of this structure is that it does not support flexible data access, because data can be accessed only by following the path down the tree structure.

Advantages of hierarchical database model

- It is the easiest model of database.
- It is secure model as nobody can modify the child without consulting to its parent.
- Searching is fast.
- Very efficient in handling one- to- many relationship.

Disadvantages hierarchical database model

- It is old fashion, outdated database model
- Modification and addition of child without consulting its parent is impossible.
- Cannot handle many- to- many relationships.
- Increase redundancy.
- It does not support flexible data access, because data can be accessed only by following the path down the tree structure.

5. Network Data Model:

A network data model is an extension of the hierarchical database structure. In this model also, the data elements of a database are organized in the form of parent-child relationships and all types of relationships among the data elements must be determined when the database is first designed. In a network database, a child data element can have more than one parent element or no parent at all. Moreover, in this type of database, the database management system permits the extraction of the needed information from any data element in the database structure, instead of s3tarting from the root data element.

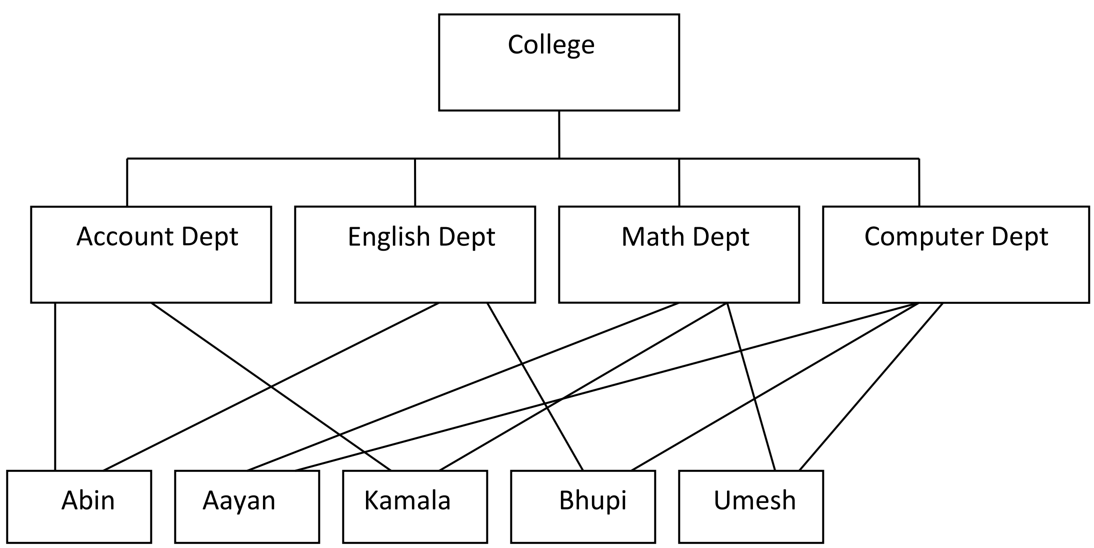


Fig: An example of a network database

Advantages of network database model

- More flexible than hierarchical database because it accept many to many relationship.
- Searching is faster because of multidirectional pointers.

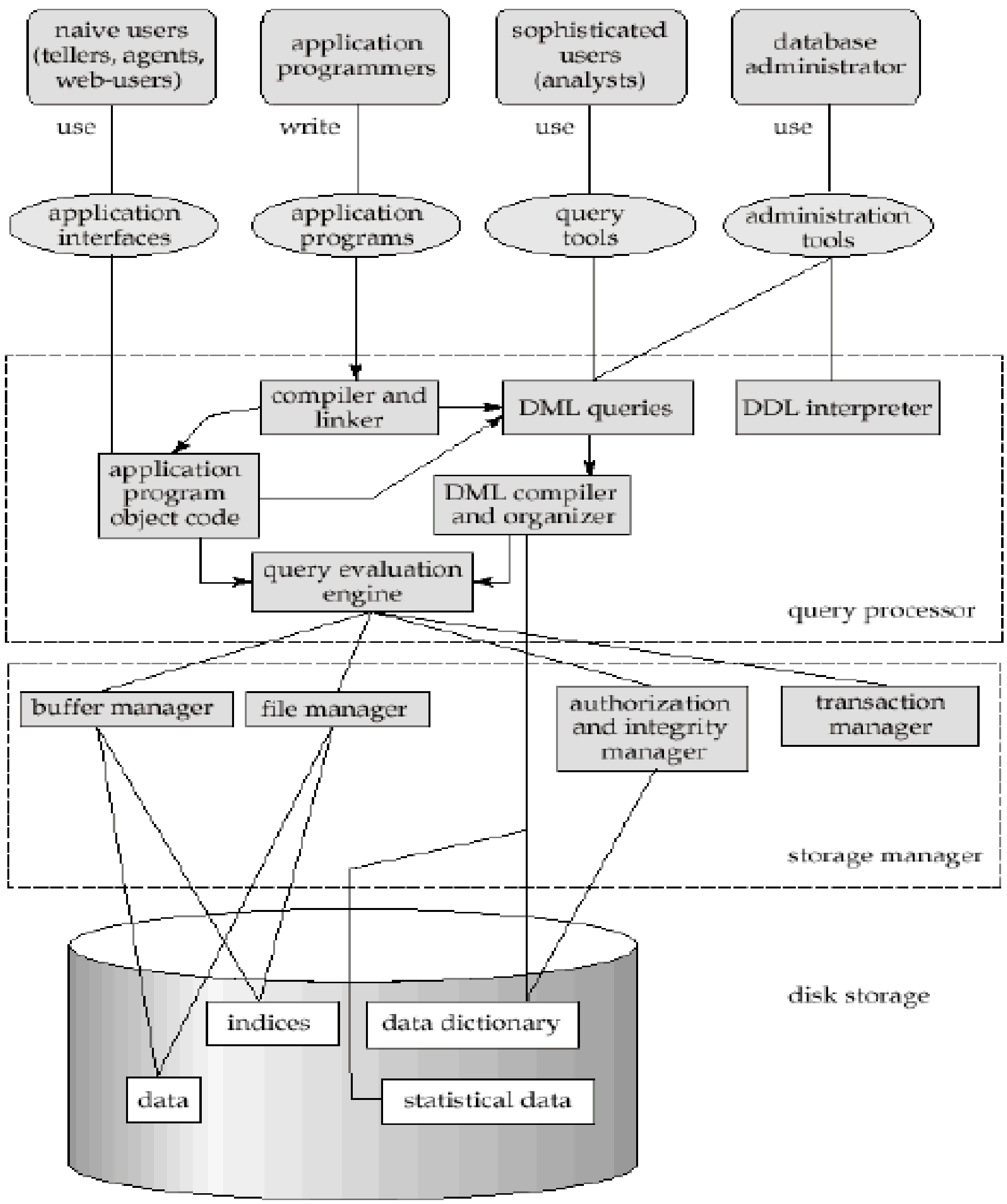
- Promotes database integrity
- Data independence

Disadvantages of network database model

- Less secure than hierarchical as it is open to all.
- Need long program to handle the relationship.
- Pointer is used in this database and that increased the overhead of storages
- Lack of structural independence

Database System Structure:

The functional components of a database system can be broadly divided into the **storage manager** and the **query processor** components.



Storage Manager:

The storage manager translates the various DML statements into low-level file-system commands. Storage manager is responsible for storing, retrieving, and updating data in the database. The storage manager components include:

- ✓ **Authorization and integrity manager**: tests for the satisfaction of integrity constraints and checks the authority of users to access data.
- ✓ **Transaction manager**: ensures that the database remains in a consistent state despite system failures, and the concurrent transaction executions proceed without conflicting.
- ✓ **File manager**: manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
- ✓ **Buffer manager**: responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

Disk Storage:

- ✓ **Data files**: which store the database itself.
- ✓ **Data dictionary**: which stores metadata about the structure of the database, in particular the schema of the database.
- ✓ **Indices**: which provides fast access to data items that hold particular values.

***The Query Processor*:**

The query processor components are:

- ✓ **DDL interpreter**: interprets DDL statements and records the definitions in the data dictionary.
- ✓ **DML compiler**: translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
- ✓ **Query evaluation engine**: which executes low-level instructions generated by the DML compiler.

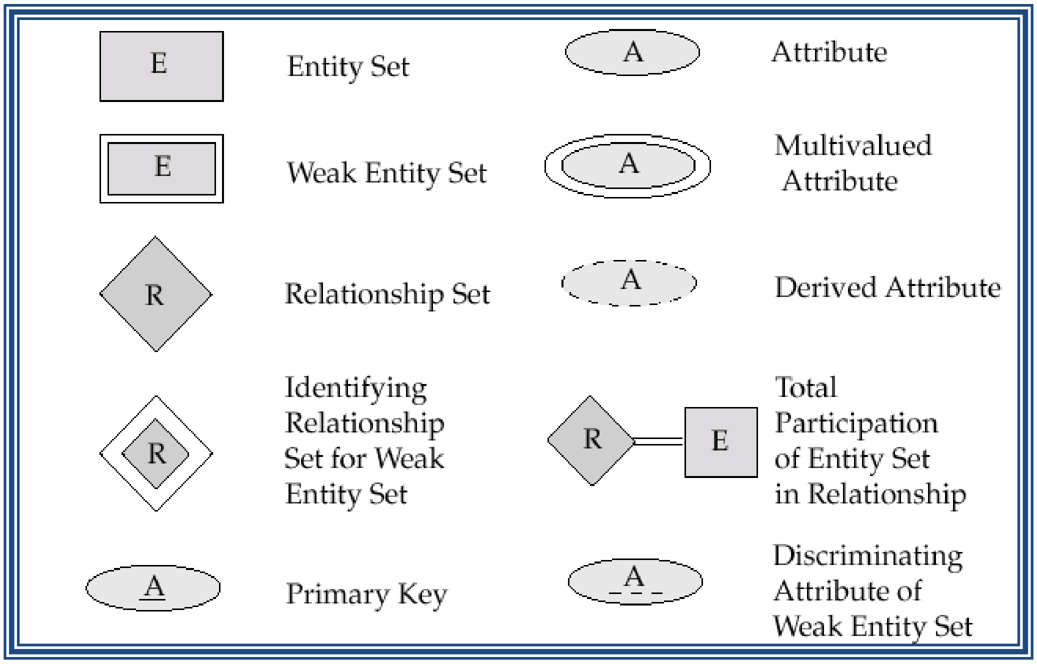
Unit 2 Entity-Relationship Model

- Basic concepts
- Mapping constraints
- Key
- Entity-relationship diagram
- Weak entity set
- Extended E-R features
- Reduction of an E-R schema to tables

The E-R data models is based on a perception of real world that consist of a collection of basic objects called entities and relationship among these objects. In an E-R model a **database** can be modeled as a collection of ***entities, and relationship*** among entities.

Notation of E-R diagram:

- ✓ **Rectangles** represent entity sets.
- ✓ **Diamonds** represent relationship sets.
- ✓ **Lines** link attributes to entity sets and entity sets to relationship sets.
- ✓ **Ellipses** represent attributes
- ✓ **Double ellipses** represent multivalued attributes.
- ✓ **Dashed ellipses** denote derived attributes.
- ✓ **Underline** indicates primary key attributes
- ✓ **Double Lines** indicate total participation of an entity set in a relationship set.
- ✓ **Double Rectangles** represent weak entity sets.
- ✓ **Double Diamonds** represent identifying relationship set for weak entity set.



Entity:

An **entity** is an object that exists and is distinguishable from other objects. An entity may be has a set of properties and the values for some set of properties may uniquely identifies an entity. For example: specific person, specific company, event, a particular plant etc. The entities have *attributes, for e*xample people have *names* and *addresses.*

Eg: A particular person

Entity Set:

An **entity set** is a set of entities of the same type that share the same properties or attributes. For example: set of all persons who are customers at a particular Bank can be defined as the entity set customer.

Eg:

Customer

Customer_id	Customer_name	Customer_street	Customer_city
1	Bhupi	Chandani	Kanchanpur
2	Arjun	Balkhu	Kathmandu
3	Abin	Pulchoak	Lalitpur

Entity set

Entity

Attributes:

The properties or characteristics of an entity are called **attributes**. For example, a *customer* entity can have *customer-id*, *customer-name*, *customer-street*, and *customer-city* as attributes.

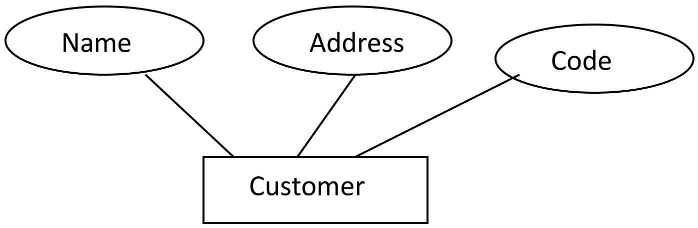


Fig: Entity with attributes

Attribute Types

An Attribute, as used in E-R model, can be characterized by the following attributes types:

Simple and composite attribute:

- **Simple:** The attributes that cannot be divided into subparts (ie. Into attributes) are called simple attributes. For example roll-number attribute of a student cannot be further divided into sub parts thus roll-number attribute of a student entity acts as a simple attribute.
- **Composite:** The attributes that can be divided into subparts (ie into attributes) are called composite attributes. For example name attribute of a particular student can be further vided into subparts first-name, middle-name, and last-name thus name attribute acts as a composite

.

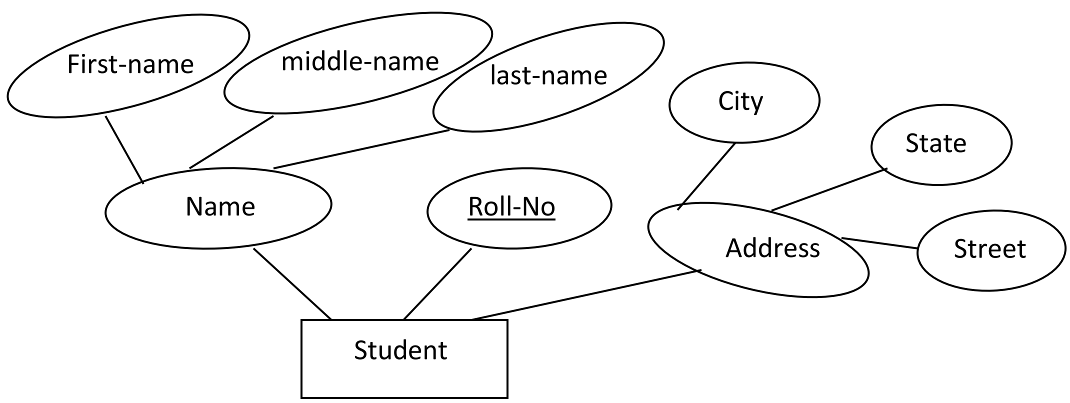
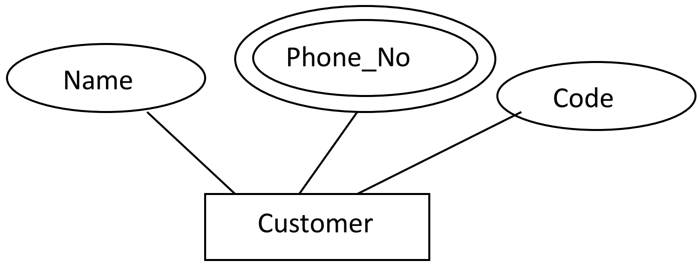


Fig:- Simple and composite attributes of Student entity

Single-valued and multi-valued attributes:

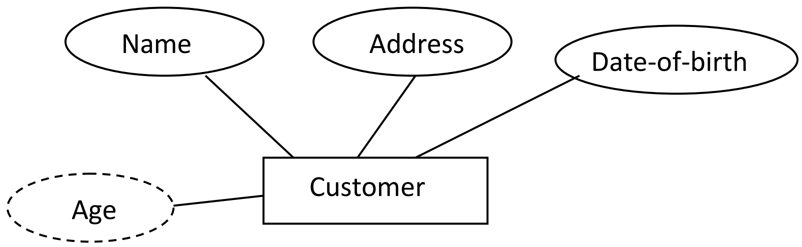
- **Single-valued:** The attributes which has a single value for a particular entity is called single-valued attributes. For example almost of our example has the single value attributes; loan-number specifies loan entity refers only one lone number.

- **Multi-valued:** If an attribute has a set of value for a specific entity is called multi-valued attributes. For example: multi-valued attribute: 'phone_number' of an employee may have zero, one or several phone numbers.



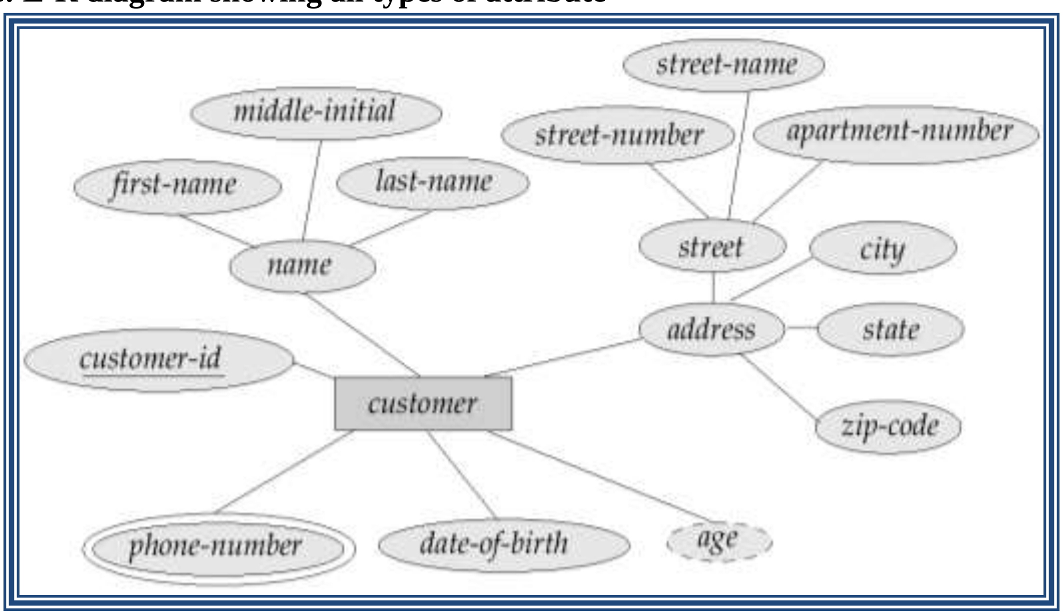
- **Derived attributes:**

The attribute whose value derived from the values of other related attributes or entities is called derived attribute. For example: age, given date_of_birth.



Note: All attributes take a **null** value when an entity does not have a value for it. The null value may indicate "not applicable", that is, the value does not exist for the entity.

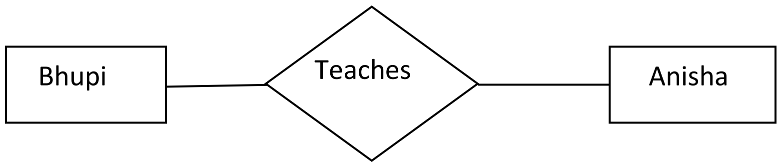
Example: E-R diagram showing all types of attribute



Relationship and Relationship sets:

A **relationship** is an association among two or more entities.

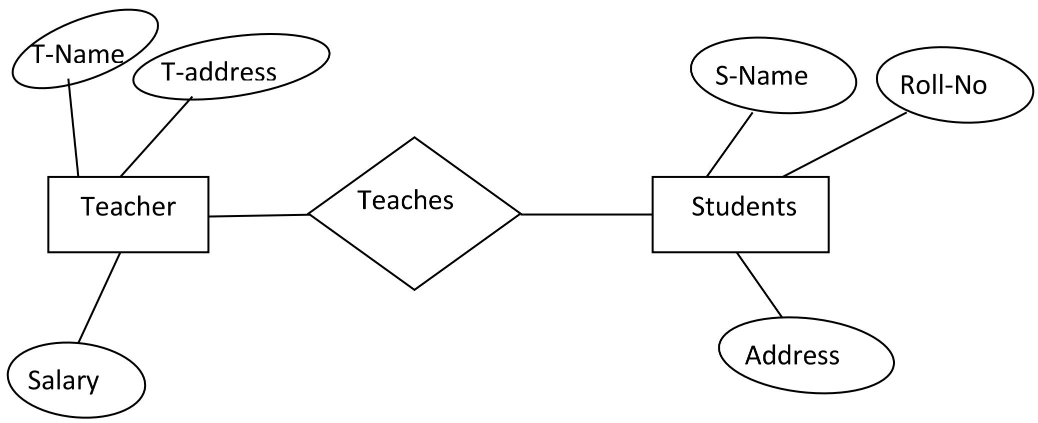
For example we may define a relationship which associates the teacher "Bhupi" with a student of name "Anisha". This specifies that Bhupi is a teacher who teaches a student of name "Anisha".



A **relationship set** is a set of relationships of the same type.

Formally, it is a mathematical relation on $n \geq 2$ entity sets. If $E_1, E_2, \ldots, E_n$ are entity sets, then a relationship set R is a subset of $\{(e_1, e_2, \ldots, e_n) | e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$, where $(e_1, e_2, \ldots.., e_n)$ is a relationship.

For example: Teacher teaches students



A relationship set may also have attributes called ***descriptive attributes***. For example, the *teach* relationship set between entity sets *teacher* and *students* may have the attribute *teaches-date*.

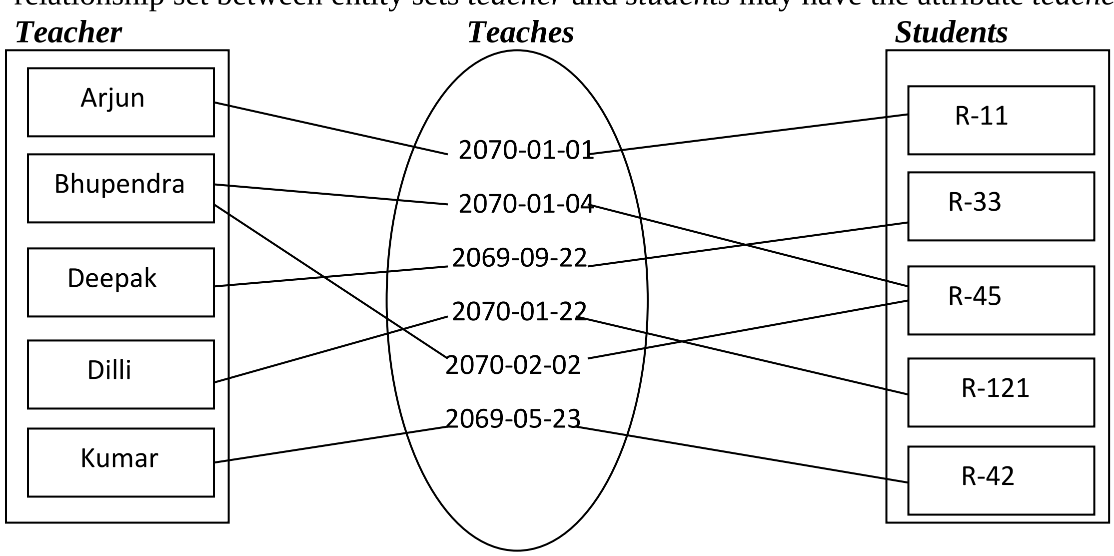
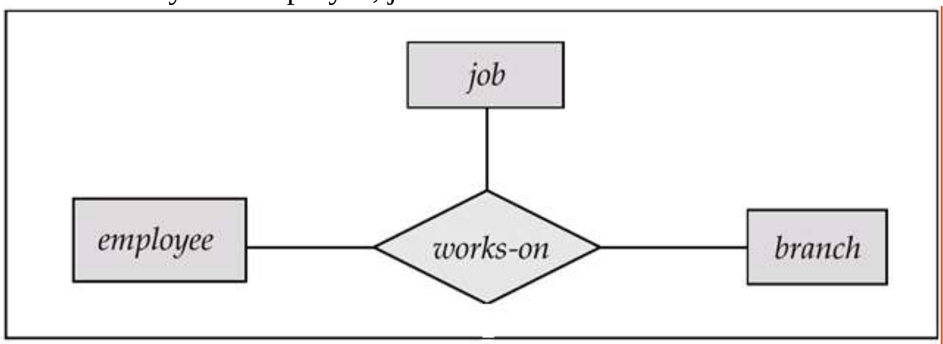


Fig: Showing relationships with descriptive attributes

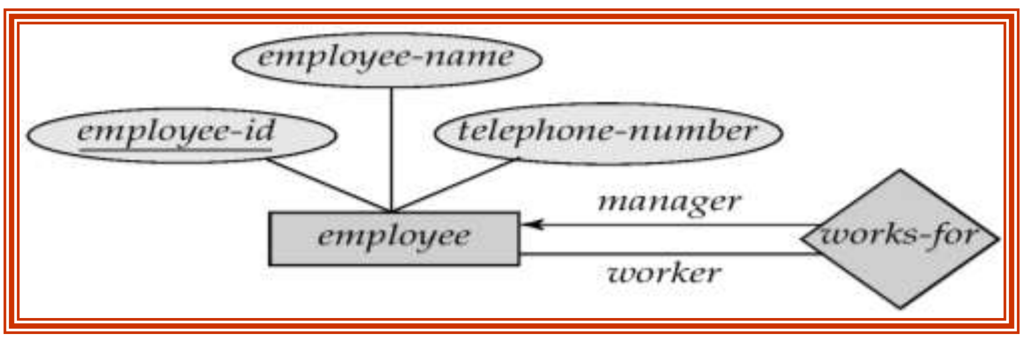
Degree of a relationship:

Degree of a relationship set refers to the number of entity sets that participate in a relationship set. Relationship sets that involve two entity sets are called **binary relationship** sets. Most relationship sets in a database system are binary.

Relationship sets may involve more than two entity sets called **n-ary** relationship sets but are rarer. For example, suppose employees of a bank may have jobs (responsibilities) at multiple branches, with different jobs at different branches. Then there is a ternary relationship set between entity sets employee, job and branch.



When the entity sets of a relationship set are not distinct (ie. The same entity set participates in a relationship set more than once, in different roles). This type of relationship set is sometimes called a **recursive** relationship set.



Constraints:

An entity relationship model may define certain constraints to which the contents of a database must conform. The most important constraints are: ***mapping cardinalities*** and ***participation constraints.***

1. Mapping Cardinality Constraints:

Mapping cardinality or cardinality ratio express the number of entities to which another entity can be associated via a relationship set. The mapping cardinality is most useful in describing binary relationship sets. (mapping cardinality also used for other relationship that is ternary etc.) For a binary relationship set the mapping cardinality must be one of the following types:

- One-to- one
- One-to- many
- Many-to-one
- Many-to-many

One-to-One:

An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. The following fig shows one to one mapping cardinality of entity sets A and B.

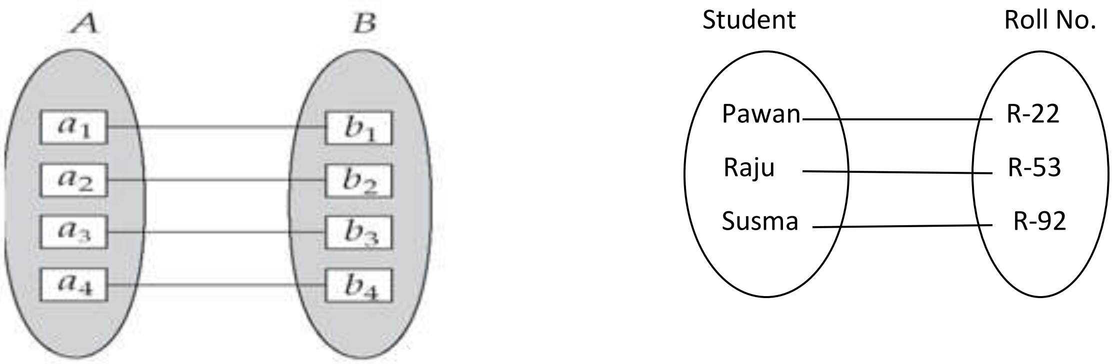


Fig: One-to-One

One-to-Many:

An entity in A is associated with any number (zero or more) of entities in B. An entity in B however can be associated with at most one entity in A. The following fig shows one-to-many mapping cardinality of entity sets A and B.

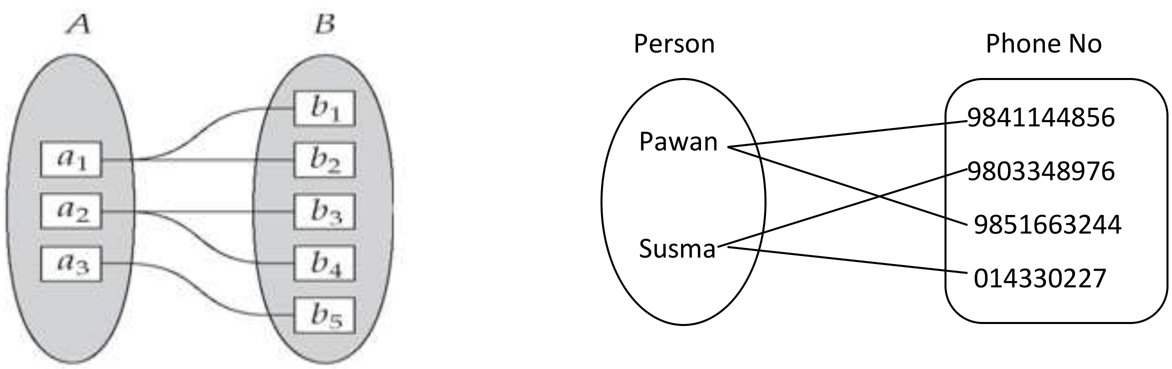


Fig: One-to-Many

Many-to-One:

An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A. The following fig clearly shows the many to one cardinality between entity sets A and B.

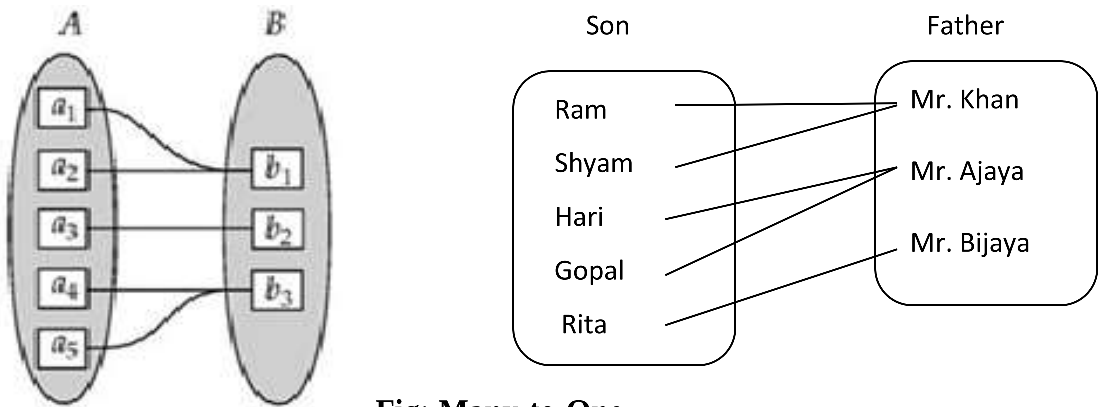


Fig: Many-to-One

Many-to-Many:

An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A. The following fig clearly shows the many-to-many cardinality between entity sets A and B.

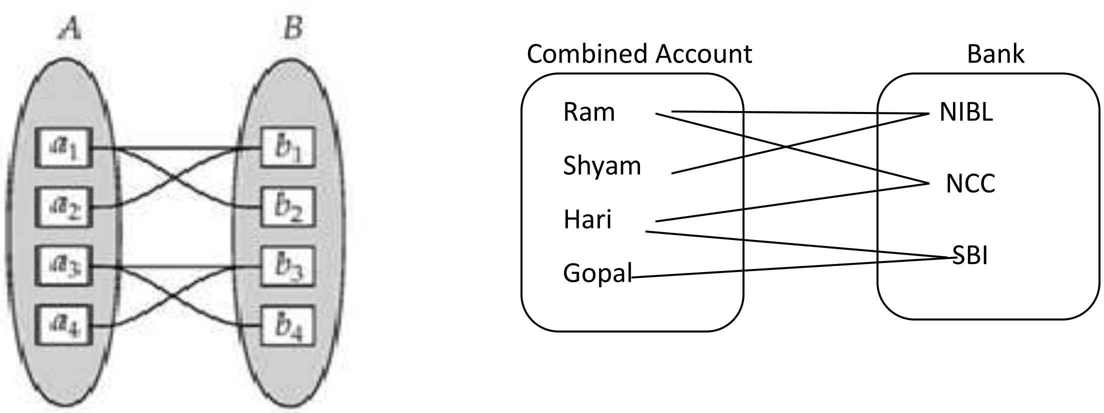


Fig: Many-to-Many

2. Participation Constraints:

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.

There are two types of participation constraints:

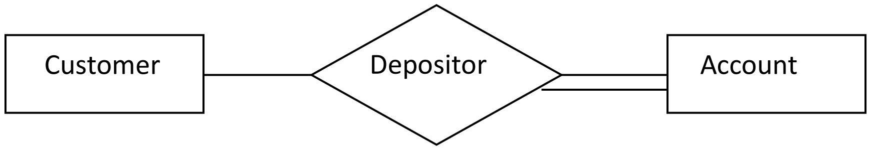
I. ***Total Participation Constraints***

II. ***Partial Participation Constraints***

Total Participation Constraints:

The participation of an entity set **A** in a relationship set **R** is said to be **total** if every entity in **A** participates in at least one relationship in **R**. For example, consider customer and account entity sets in a banking system, and a relationship set depositor between them indicating that each

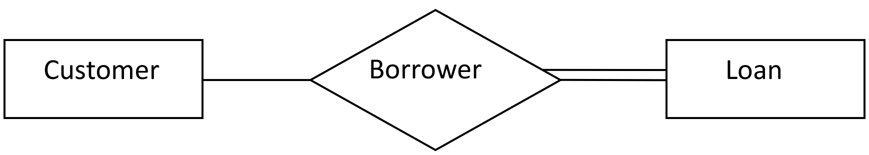
customer must have an account. Then there is total participation of entity set account in the relationship set depositor.



Note: Double lines are used to indicate that the participation of an entity set in a relationship set is total. And single lines are used to indicate that the participation of an entity set in a relationship set is partial.

Partial Participation Constraints:

If only some entities in **A** participate in relationships in **R**, the participation of entity set **A** in relationship set **R** is said to be **partial**. For example, consider customer and loan entity sets in a banking system, and a relationship set borrower between them indicating that some customers have loans. Then there is partial participation of entity set customer in the relationship set borrower.



Keys:

Keys are used to distinguish the entities within a given entity set. Keys also help to uniquely identify relationships.
There are different types of keys which are as

- Super key
- Candidate key
- Primary key
- Foreign key

Super Key:

A **super key** of an entity set is a set of one or more attributes whose values uniquely determine each entity in the entity set. If K is a super key and any superset of K is also super key, thus the concept of superset is not sufficient for our purpose.
For example Roll-No attribute of the entity set student is sufficient to distinguish one student entity from another. Thus the Roll-No is a super key. Similarly the combination of Roll-No and Name is a super key of the entity set student.

Candidate Key:

A candidate key of an entity set is a minimal super key. That is a super key which does not have any proper subset is called candidate key. For example c*ustomer-id* is candidate key of

customer, suppose the combination of customer-name and customer-street is also sufficient to distinguish among members of customer entity set. Then, both {customer-id} and {customer-name, customer-street} are candidate key.

> Note: Any candidate keys other than the one chosen as a primary key is known as ***alternate key.***

Primary key:

A primary key is a candidate key that is chosen by the database designer as the principle means of uniquely identifying entities within an entity set. There may exist several candidate keys, one of the candidate keys is selected to be the primary key.

Any two individual entities in the set are prohibited from having the same value on the key attributes at the same time (i. e values of key attributes must be unique)

For examples- the candidate key { RollNumber} can be considered to be a primary key for the entity set student. The candidate key {Customer-id} can be considered to be a primary key of customer entity set.

> Note: If a primary key contains more than one attribute then it is called **composite Key**

Foreign key:

A foreign key (FK) is an attribute or combination of attributes that is used to establish and enforce a link between the data in two tables. You can create a foreign key by defining a FOREIGN KEY constraint when you create or modify a table.

In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table.

Student

S-ID	***Name***	***Address***	***Course-ID***
S-12	Pawan	Joshi	C002
S-14	Yamman	Karki	C021
S-51	Abin	Saud	C321
S-11	Binak	Singh	C112

Foreign Keys

Relationships

Course

Course-ID	***Course-Name***
S-12	C++
S-14	DBMS
S-51	Account
S-11	Banking

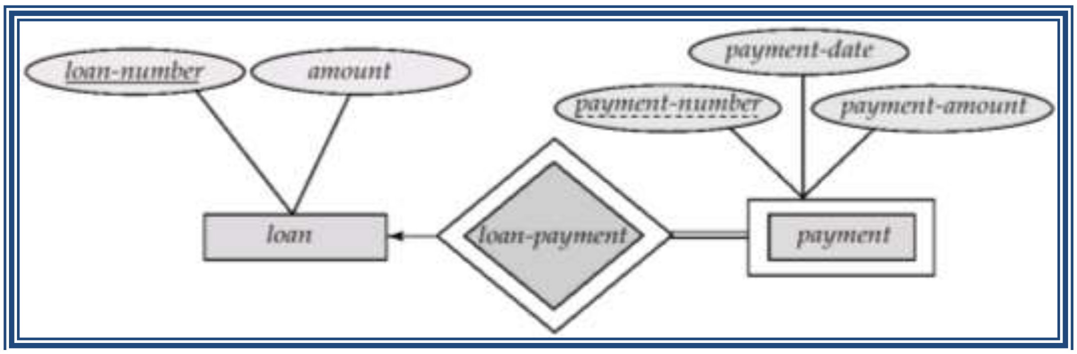
Primary Keys

Fig: Primary key and foreign key

> Note: A ***FOREIGN KEY*** in one table points to a ***PRIMARY KEY*** in another table.

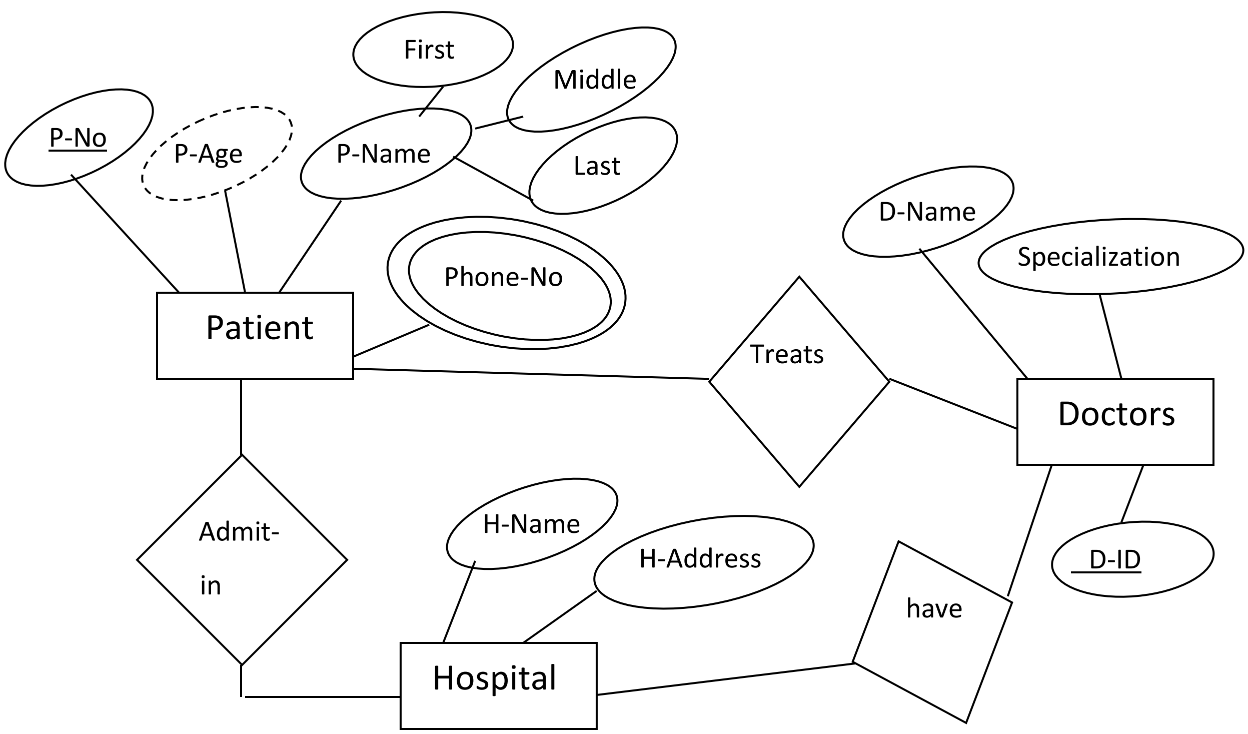
Weak entity set:

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed as a weak entity set. An entity set that has a primary key is termed as a strong entity set. For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set, using one of the key attribute of owner entity set. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship. An attribute of weak entity set that is use in combination with primary key of the strong entity set to identify the weak entity set uniquely is called discriminator (partial key).



In the above figure, *payment-number* is partial key and (*loan-number*, *payment-number*) is primary key for *payment* entity set.

Example: E-R diagram for hospital with a set of patients and medical doctors.



Extended E-R model (EER model):

The EER model includes all of the concepts introduced by the ER model. Additionally it includes the concepts of a subclass and super class, along with the concepts of specialization and generalization.

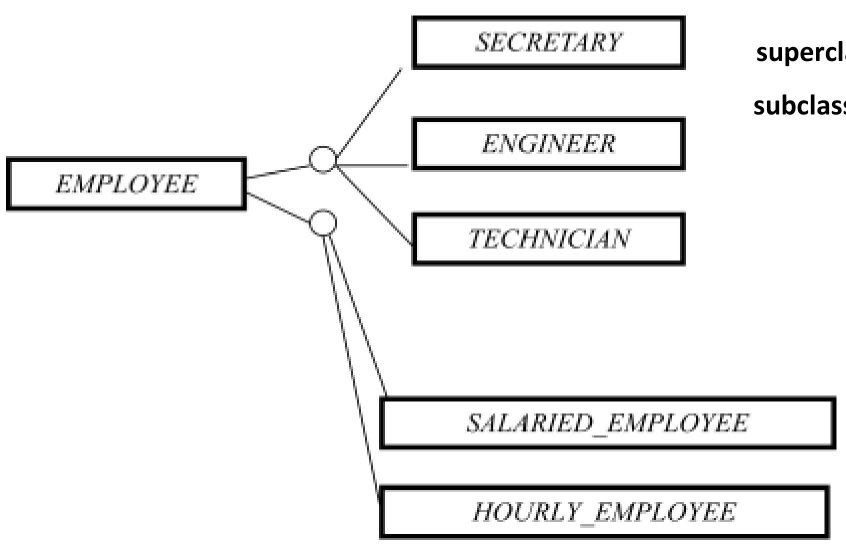
There are basically four concepts of EER-Model:

- Attribute Inheritance (subclass / superclass relationship)
- Specialization
- Generalization
- Categories
- Aggregation

Subclass and superclass:

The class that is derived from another class is called a ***subclass***. The class from which a subclass derives is called the ***superclass***. The following figure illustrates these two types of classes:

- ✓ An entity type may have additional meaningful sub-groupings of its entities. Example: EMPLOYEE may be further grouped into {SECRETARY, ENGINEER, TECHNICIAN, MANAGER, MANAGER, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE …}
- ✓ EER diagrams extend ER diagrams to represent these additional sub-groupings, called ***subclasses* or *subtypes.*** Each of these subgroups is called a **subclass** of the EMPLOYEE entity type.
- ✓ The EMPLOYEE entity type is called the ***superclass*** of each of these subclasses.
- ✓ The relationship between a ***superclass*** and any one of its subclasses is called a **superclass/subclass** or **class/subclass** or **IS-A (IS-AN) relationship** (e.g. EMPLOYEE/ SECRETARY EMPLOYEE/MANAGER).
- ✓ Subclass entities have their own specific attributes. They also inherit all attributes and relationships of its ***superclass*** (subclasses can be considered as separate entity types).



superclass: EMPLOYEE

subclasses: SECRETARY, ENGINEER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE

Fig: superclass/subclass relationship

Specialization:

The process of defining a set of subclasses from a superclass is known as specialization.
The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass. It is a **top-down design** process.
Example: {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon *job type.*

Note: There may have several specializations of the same superclass.

Generalization:

It is a **bottom-up design** process. Here, we combine a number of entity sets that share the same features into a higher-level entity set. The original classes become the subclass of the newly formed generalized superclass. The reason, a designer applies generalization is to emphasize the similarities among the entity sets and hide their differences. Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way. The terms specialization and generalization are used interchangeably.

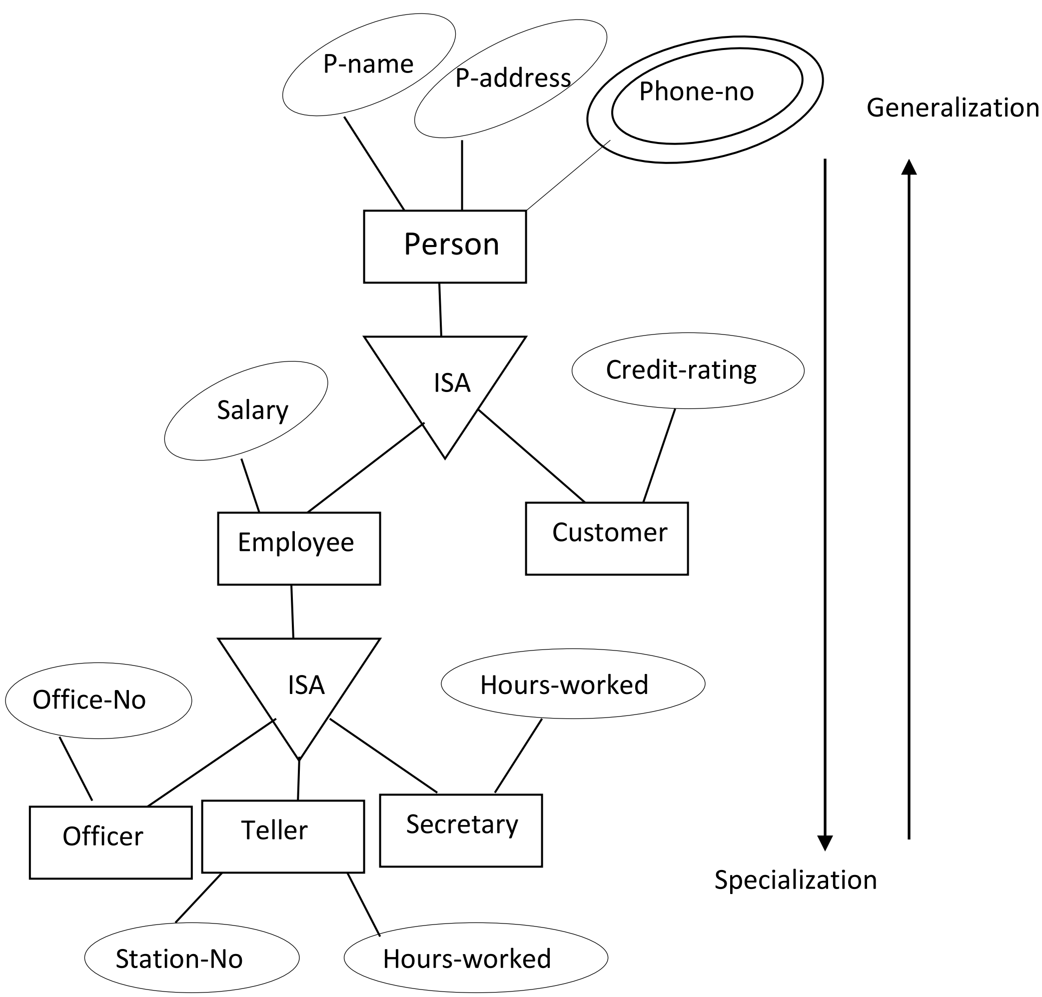


Fig: ER diagram with specialization and generalization

Note:

- In EER disjoint-constraint is illustrated by placing the letter **d** inside the circle
- In case **overlap** between subclasses is allowed, we place the letter **o** inside the circle.

Constraints on Specialization and Generalization:

Two basic constraints can apply to a specialization/generalization:

- **Disjointness Constraint:**
- **Completeness Constraint:**

Disjointness/Overlapping Constraints:

It specifies that the subclasses of the specialization must be ***disjoint***. Here an entity can be a member of at most one of the subclasses of the specialization and it is represented by ***d*** in EER diagram.

If not disjoint, specialization is ***overlapping*****.** That is the same entity may be a member of more than one subclass of the specialization and it is represented by ***o*** in EER diagram.

Completeness Constraint:

Total participation constraint specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization. It is represented by ***double line*** in EER diagram.

Partial participation constraint allows an entity not to belong to any of the subclasses and shown in EER diagrams by a ***single line***.

Aggregation:

One of the limitations of E-R model is that it cannot express relationship among relationships. One of the solutions in such a situation is using aggregation. Aggregation is an abstraction in which relationship sets (along with their associated entity sets) are treated as higher-level entity sets and can participate in relationships.

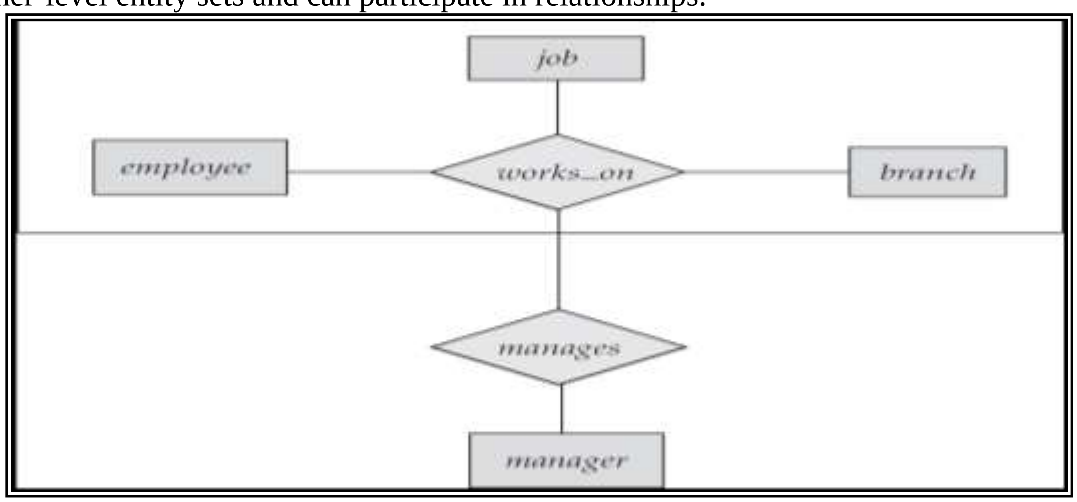


Fig: E-R Diagram with Aggregation

Reduction of E-R Schema to tables:

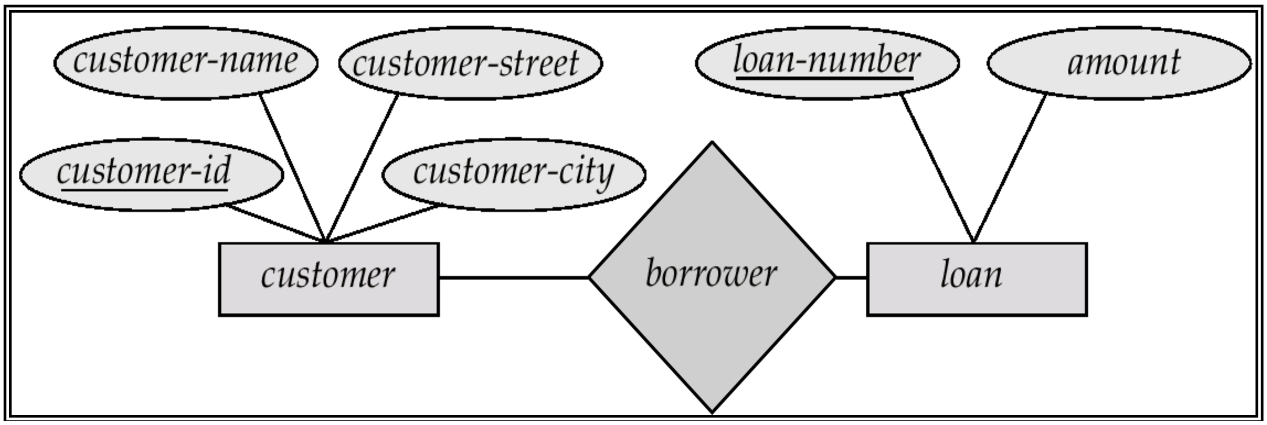
To reduce given ER diagram into table simply we create a table for each entity set and for each relationship sets. And that assigned the name of the corresponding entity set or relationship set as table name. Generally the number of attributes of an entity set or relationship set equal to the degree of a corresponding table (fields of a table).

To reduce given ER diagram into tables normally we divide ER diagram into three sections:

- Strong entity sets
- Weak entity sets and
- Relation sets
-

Reducing strong entity sets into tables:

Consider an E-R diagram as given below



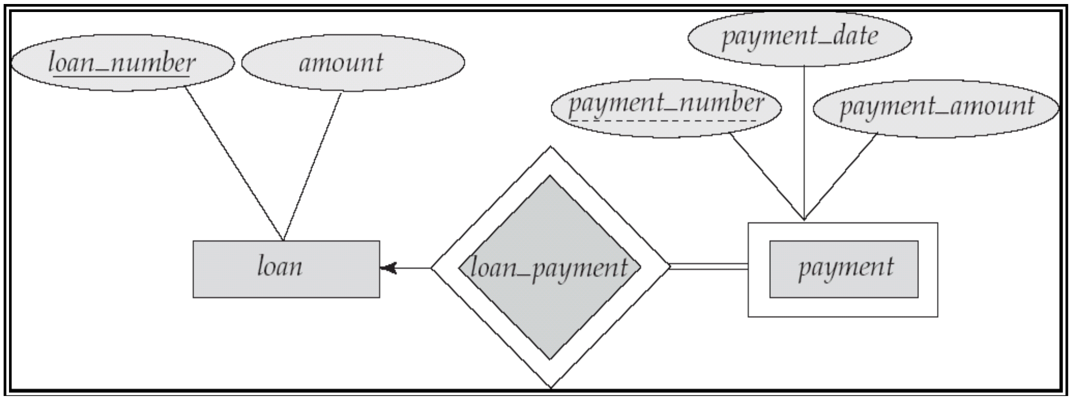
The tabular representation of the entity set loan of the given E-R diagram, This entity has two attributes loan-number and amount. We represent this entity set by a table called loan, with two columns named loan-number and amount as below:

Loan

loan_number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Reducing weak entity sets into tables:

To illustrates this consider the entity set payment in the following E-R diagram



The table of entity '*payment*' consist the column names ***loan-number, payment-number, payment-data, and payment-amount as below:***

Payment

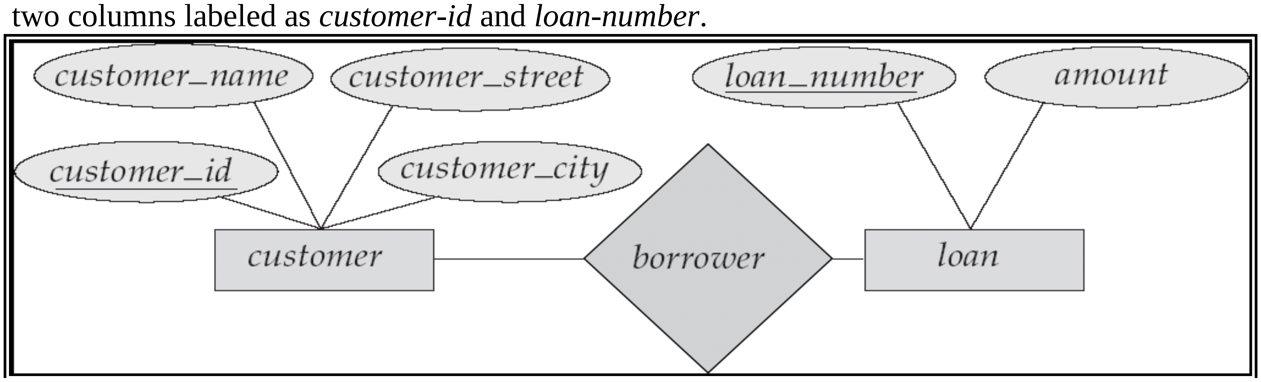
Loan-No	**Payment-No**	**Payment-Date**	**Payment-amount**
L-11	L-11	2069-02-22	50,000
L-22	L-22	2069-04-28	70,000
L-07	L-07	2069-01-19	45,000
L-32	L-32	2070-02-02	98,000

Reducing Relationship sets into tables:

To explain this, consider a relationship set ***borrower*** in E-R diagram and this relationship set involves the following entity sets:

- ***Customer*** with the primary key *customer-id*
- ***Loan*** with the primary key l*oan-number*.

This relationship set does not have any its own descriptive attributes, so the ***borrower*** table has two columns labeled as *customer-id* and *loan-number*.

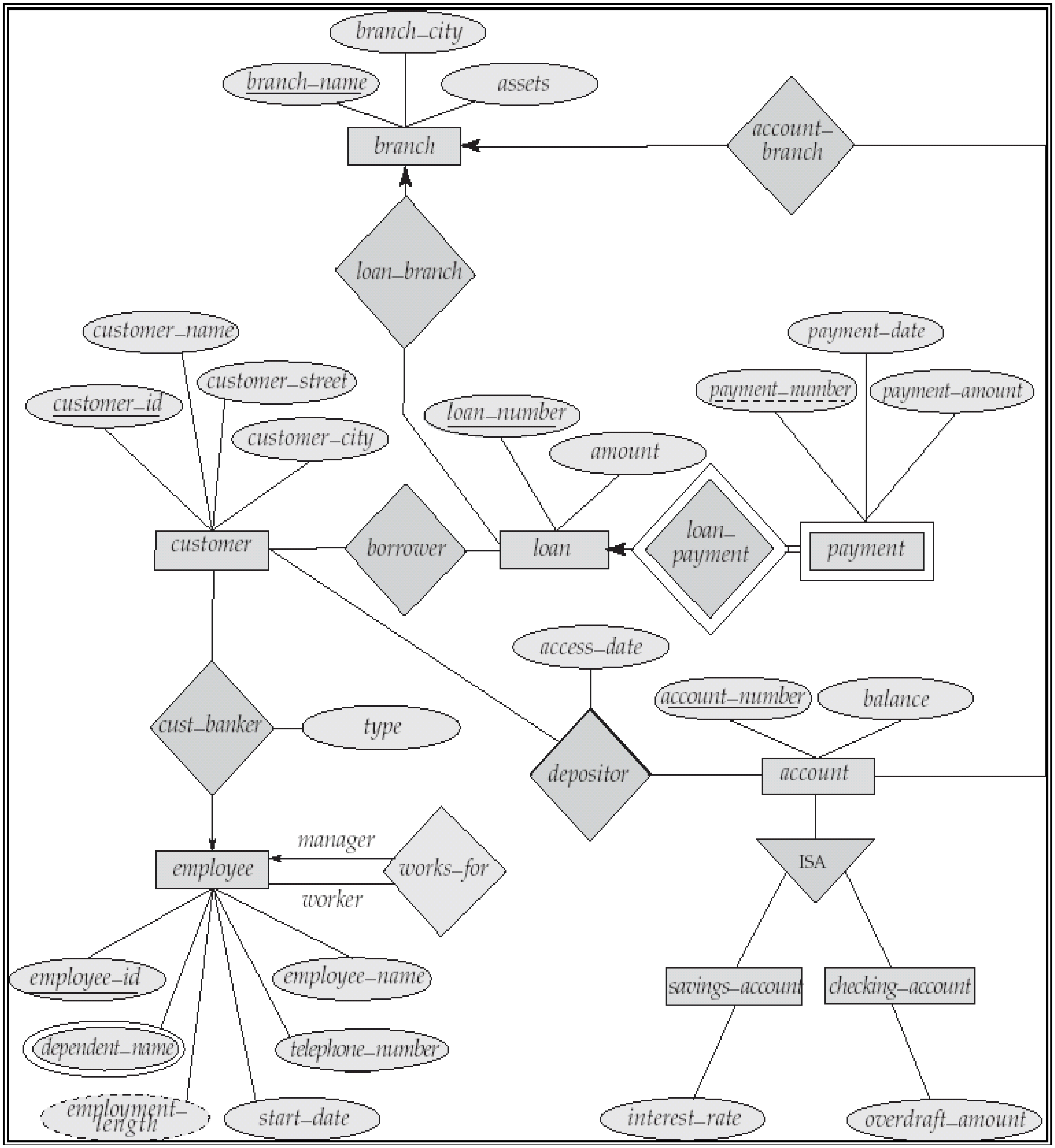


Borrower

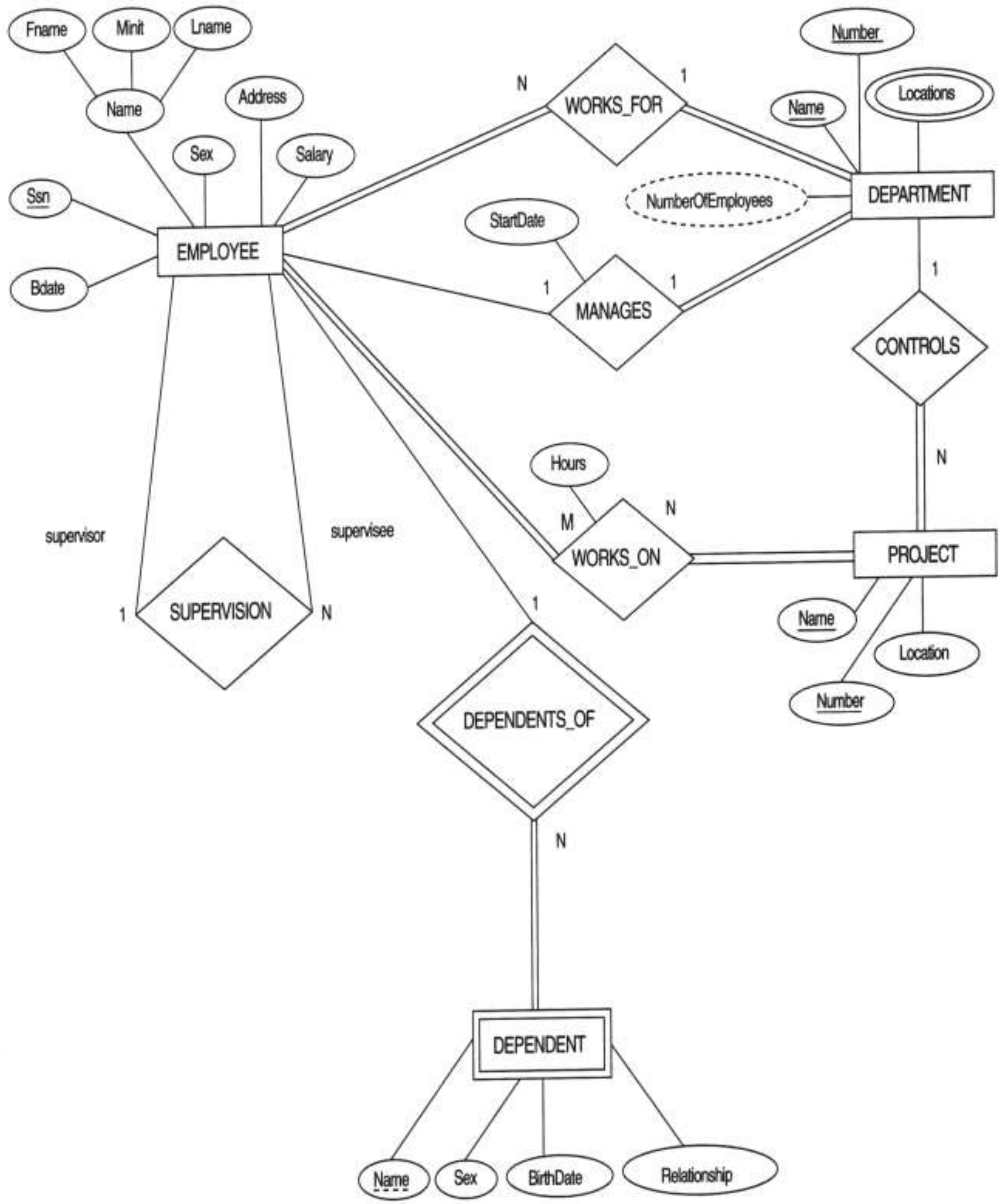
Customer-id	***Loan-number***
019-29-3746	L-11
019-28-3123	L-17
019-24-3144	L-76

E-R Diagram for a Banking Enterprise

The E-R diagram for the banking enterprise is given below.



Example: ER diagram for Company database system



Example 3: Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

Or

patients (patient-id, name, insurance, date-admitted, date-checked-out)
doctors (doctor-id, name, specialization)
test (testid, testname, date, time, result)
doctor-patient (patient-id, doctor-id)
test-log (testid, patient-id) performed-by (testid, doctor-id)

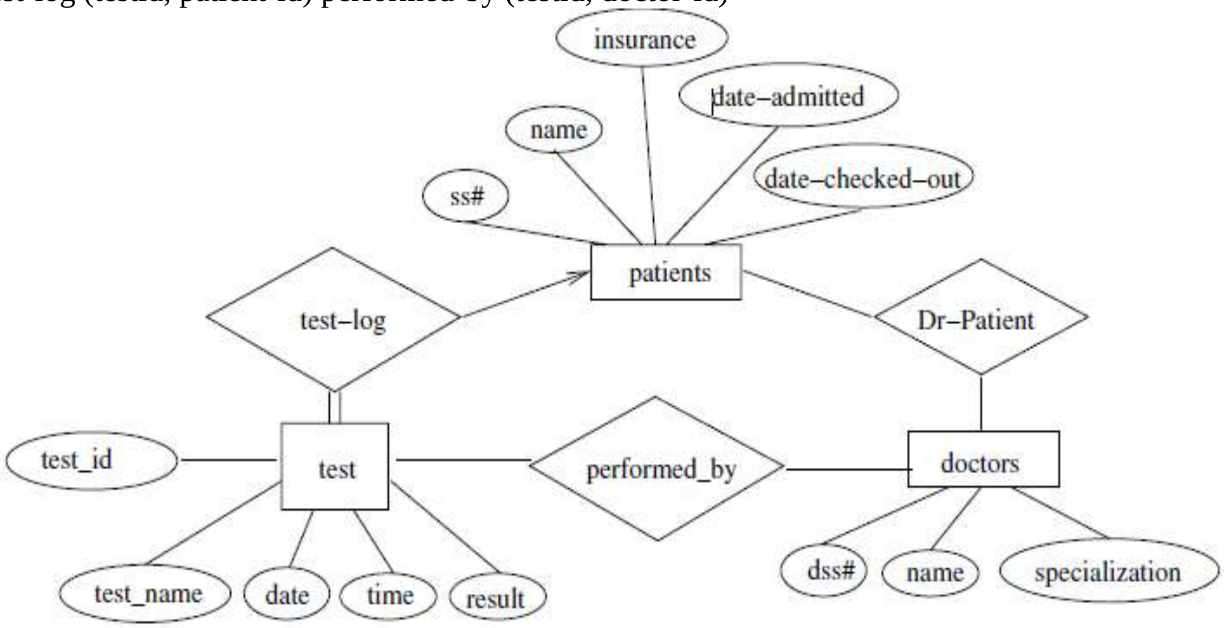


Figure E-R diagram for a hospital.

Example 4 Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

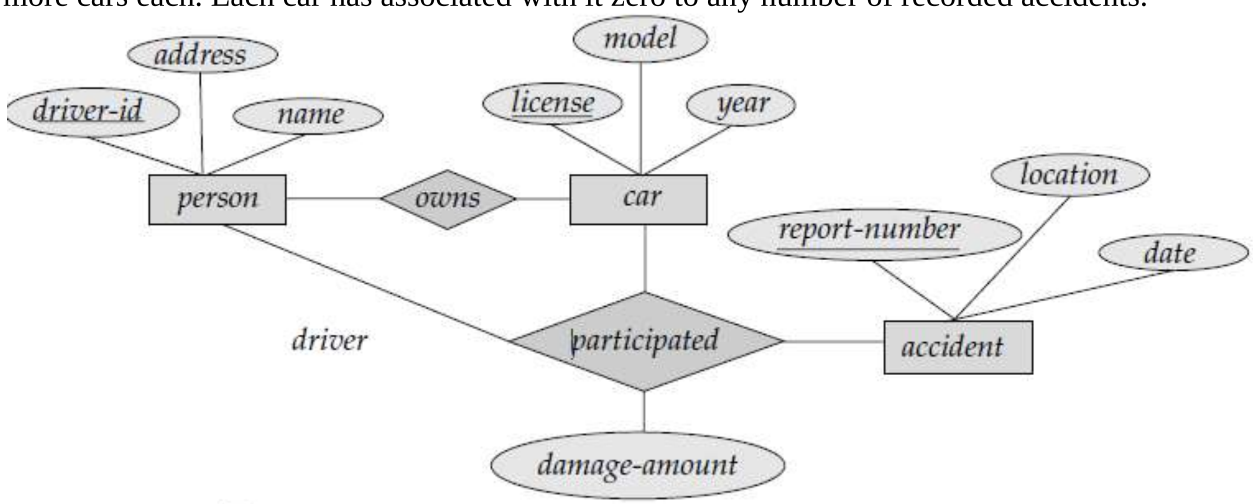


Figure E-R diagram for a Car-insurance company.

Unit 3

Relational Model

- Structure of a relational database
- The Relational Algebra
  - ✓ Select, Project, Product, Union, Difference, Rename, Intersection, Division, Assignment, natural Join, Outer Join, Aggregate functions, generalized projection.
  - ✓ Database Manipulation: Insertion, deletion, updates.

Relational Model:

The First database systems were based on the network and hierarchical models. The relational model was first proposed by E.F. Codd in 1970 and the first such system (notably INGRES and System/R) was developed in 1970s. The relational model is now the dominant model for commercial data processing applications.

Structure of relational databases:

A relational database consists of a collection of **tables**, each having a unique **name**. A row in a table represents a **relationship** among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of a **table** and the mathematical concept of a **relation**.

Basic Structure

Figure shows the **deposit** and **customer** tables for banking example.

Deposit

b-name	account#	cname	balance
Ktm- branch	101	Bipin	500
Lalitpur-branch	215	Anisha	700
Kirtipur-branch	102	Abin	400
Pokhara-branch	304	Binita	1300

Customer

cname	street	ccity
Arjun	Pender	Vancouver
Bhupi	kumariclub	Burnaby
Ajaya	newroad	kathmandu
Rahul	No.3 Road	Richmond
Umesh	Chandani	Mahendranagar

Figure: The deposit and customer relations.

- Relation deposit has four attributes.
- For each attribute, there is a permitted set of values, called the **domain** of that attribute. E.g. the domain of *b-name* is the set of all branch names (ie Ktm-branch, Lalitpur-branch, Kirtipur-branch, pokhara-branch).
- Let D1 denote the domain of *bname*, and D2, D3 and D4 the remaining attributes' domains respectively.
  Then, any row of *deposit* consists of a four-tuple (v_1,v_2,v_3,v_4) where
  $\{V1 \in D1, V2 \in D2, V3 \in D3, V4 \in D4\}$
- In general, *deposit* contains a **subset** of the set of all possible rows.

That is, *deposit* is a subset of
D1 X D2 X D3 X D4

Query languages:

- A query language is a language in which a user requests information from the database. Query languages can be categorized as either procedural or non-procedural. In a ***procedural language*** the user instructs the system to perform a sequence of operations on the database to compute the desired result. Example: ***Relational algebra***.
- In a ***non-procedural language***, the user describes the desired information desired without giving a specific procedure for obtaining that information. Example: tuple relational calculus, domain relational calculus, SQL etc.

Relational database management system:

A **relational database management system** (**RDBMS**) is a database management system (DBMS) that is based on the relational model. An important feature of relational systems is that a single database can be spread across several tables. This differs from flat-file databases, in which each database is self-contained in a single table.

The Relational algebra:

A relational algebra is a collection of formal operations acting on relations and producing relations as result. It is one of the procedural query language in which a user requests information from a database. The main operations of the relational algebra are the set operations (such as union, intersection and Cartesian product), selection (keeping only some lines of a table) and the projection (keeping only some columns).

Operations in Relational Algebra:

1. Fundamental operations:

- *Select operation*
- *Project operation*
- *Union operation*
- *Set difference operation*
- *Cartesian product operation*
- *Rename operation*

2. Additional operations:

- *Set-intersection operation*
- *Natural-join operation*
- *Division operation*
- *Assignment operation*

3. Extended Relational Algebra Operations

- *Generalized projection*
- *Aggregate functions*
- *Outer join*
- *Null values*

1. Fundamental operations:

1.1 Select operation:

The selection operation is used to extract tuples (rows) from a relation that satisfy a given predicate. It is denoted by sigma symbol (σ).

Syntax: - σ *<condition> (Relation)*

Selection Example:

Assume the following relation Employee has the following tuples:

Employee

Name	Office	Dept	Rank
Bipin	400	Computer	Assistant
Niky	220	Economics	Adjunct
Rahul	160	Economics	Assistant
Binita	420	Computer	Associate
Solu	500	Finance	Associate

- Select only those Employees who involve in the Computer department:

  $\sigma_{Dept = 'Computer'}$ (Employee)

Result:

Name	Office	Dept	Rank
Bipin	400	CS	Assistant
Binita	420	CS	Associate

- Select only those Employees with first name *Solu* who are associate professors:

  $\sigma_{Name = 'Solu' \land Rank = 'Assistant'}$ (Employee)

Result:

Name	Office	Dept	Rank
Solu	400	Finance	Associate

- Select only those Employees who are either Assistant Professors or in the Economics department:

  $\sigma_{Rank = 'Assistant' \lor Dept = 'Economics'}$ (Employee)

Result:

Name	Office	Dept	Rank
Bipin	400	Computer	Assistant
Rahul	160	Economics	Assistant
Niky	220	Economics	Adjunct

- Select only those Employees who are not in the Computer department or Adjuncts:

  $\sigma_{\neg (Rank = 'Adjunct' \land Dept = 'Computer')}$ (Employee)

Result:

Name	Office	Dept	Rank
Rahul	160	Economics	Assistant
Solu	500	Finance	Associate

Exercises

Evaluate the following expressions:

1. $\sigma_{\neg \text{Rank = 'Adjunct'} \wedge \text{Dept = 'Computer')}}$ (Employee)
2. $\sigma_{\text{Rank = 'Associate'}}$ ($\sigma_{\text{Dept = 'Computer'}}$ (Employee))
3. $\sigma_{\text{Dept = 'Computer'}}$ ($\sigma_{\text{Rank = 'Associate'}}$ Employee)
4. $\sigma_{\text{Rank = 'Associate'} \wedge \text{Dept = 'Computer'}}$ (Employee)
5. $\sigma_{\text{Age > 26}}$ (R U S)

1.2 Project operation:

Projection operation is used to extracts specified columns (arity) of a relation. With the help of this operation, any number of columns can be omitted from a table or columns of table can rearrange.

Syntax: - π***<attribute-list> (Relation)***

Projection Examples:

Assume the same Employee relation above is used.

- Project only the names and departments of the employees:

$\pi_{\text{name, dept}}$ (Employee)

Results:

Name	Dept
Bipin	Computer
Niky	Economics
Rahul	Economics
Binita	Computer
Solu	Finance

Combining Selection and Projection Operations:

- The selection and projection operators can be combined to perform both operations.
- Show the names of all employees working in the 'Computer' department:

π_{name} ($\sigma_{\text{Dept = 'Computer'}}$ (Employee))

Results:

Name
Bipin
Binita

- Show the name and rank of those Employees who are not in the 'Computer' department or Adjuncts:

  $\pi_{\text{name, rank}}(\sigma_{\neg(\text{Rank = 'Adjunct'} \vee \text{Dept = 'Computer'})}(\text{Employee}))$

Result:

Name	Rank
Rahul	Assistant
Solu	Associate

Exercises

Evaluate the following expressions:

1. $\pi_{\text{name, rank}}(\sigma_{\neg \text{Rank = 'Adjunct'} \wedge \text{Dept = 'Computer'}}(\text{Employee}))$
2. $\pi_{\text{fname, age}}(\sigma_{\text{Age > 22}}(R \cup S))$

1.3 Union Operation (∪):

Consider the following relations **R** and **S.** The **union** of relations R and S is denoted by R U S and it is the set of tuples that are either in R or in S or in both. It returns the union (set union) of two compatible relations. For a union operation to be legal, we require that invoked relations must have the same number of attributes and corresponding attributes have same type.

R

First	Last	Age
Bill	Smith	22
Kamala	dhami	21
Maya	Singh	23
Anisha	Jha	22

S

First	Last	Age
Pinky	ojha	36
Maya	singh	23
Anisha	KC	22

Result: Relation with tuples from R and S with duplicates removed.

First	Last	Age
Bill	Smith	22
kamala	dhami	21
Maya	singh	23
Anisha	Jha	22
Pinky	Ojha	36
Anisha	KC	22

1.4 Set Difference Operation (-):

Set difference is denoted by the minus sign (**-**). It finds tuples that are in one relation, but not in another. Thus results in a relation containing tuples that are in **R** but not in **S**.
Result: Result: Relation with tuples from R but not from S
R - S

First	Last	Age
Bill	Smith	22
Maya	Singh	23
Anisha	Jha	22

1.5 Cartesian product(X):

The Cartesian product operation does not require relations to union-compatible i.e. the involved relations may have different schemas. The **Cartesian product** of two relations R and S is denoted by R **X** S, is the set of all possible combinations of tuples of the two relations.
Example:

R

First	Last	Age
Kamala	Ojha	22
Pawan	Bhatt	23
Anisha	KC	32

S

Dinner	Dessert
Steak	Ice Cream
Lobster	Cheesecake

Result: Produce all combinations of tuples from two relations.
R X S

First	Last	Age	Dinner	Dessert
Kamala	Ojha	22	Steak	Ice Cream
Kamala	Ojha	22	Lobster	Cheesecake
Pawan	Bhatt	23	Steak	Ice Cream
Pawan	Bhatt	23	Lobster	Cheesecake
Anisha	KC	32	Steak	Ice Cream
Anisha	CK	32	Lobster	Cheesecake

Key points to remember to ***Union Compatible*** Relations:
Two relations R and S are *union compatible* if and only if they have the same degree and the domains of the corresponding attributes are the same.

- Attributes of relations need not be identical to perform union, intersection and difference operations.
- However, they must have the same number of attributes or *arity* and the *domains* for corresponding attributes must be identical.
- **Domain** is the data type and size of an attribute.
- The **degree** of relation R is the number of attributes it contains.

1.6 Rename Operation:

The rename operator is denoted by rho (ρ).
It can be used in two ways:

- $\rho_x(E)$ return the result of expression E in the table named *x*.
- $\rho_{x(A_1,A_2,\ldots,A_n)}(E)$ return the result of expression E in the table named *x* with the attributes renamed to $A_1, A_2, \ldots, A_n$.

It is mainly used in the situation where we need to find the Cartesian product of a relation with itself i.e. Account × Account.
For that we should rename one of the account tables by some other name to avoid the confusion.

Example: $\boldsymbol{\rho}_{Emp(Name1,Dept1)}(Employee)$

Employee

Name	Department
Bhupi	IT
Arjun	CSC
Aayan	IT

After rename ⟹

Emp

Name1	Dept1
Bhupi	IT
Arjun	CSC
Aayan	IT

2. Additional operations:

2.1 Set Intersection Operation (∩):

Set intersection is denoted by symbol ∩ and it returns a relation that contains tuples that are in **both** of its argument relations.
Result: Relation with tuples that appear in both R and S.

R ∩ S

First	Last	Age
Maya	singh	23

2.2 Join operations:

- Natural join (⋈)
- Theta join ($⋈_\theta$)
- ***Outer Join***
  - Left outer join(⟕)
  - Right outer join(⟖)
  - Full outer join(⟗)

The Join operation is used to combine related tuples from two relations into single tuples.

Natural join operation (⋈):

The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the join symbol . The natural join operation performs the Cartesian product of given relations together with remove the duplicate attributes. The natural join thus performs a join by equating the attributes with the same name and then eliminates the replicated attributes.
In brief the result of the natural join of two relations R and S is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names.

Formal definition of natural join:

Let R and S be any two relations and $\{A_1, A_2, A_3, \ldots, A_n\}$ are n attributes of given relations then their natural join is denoted by R ⋈ S and is defined as follow:

$$R \bowtie S = \pi_{R \cup S} (\sigma_{R.A1 = S.A1 \wedge R.A2 = S.A2 \wedge R.A3 = S.A3 \ldots \wedge R.An = S.An}\, R \times S)$$

Where R ∩ **S = {A1, A_2, A_3.....A_n}**

For example consider the tables *Employee* and *Dept* and their natural join:

Employee

e-id	e-name	Dept
11	Bhupi	Computer
13	Anju	Finance
43	Manju	Computer
54	Nisha	Finance

Department

Dept	Manager
Computer	Anisha
Finance	Manisha
Production	Umesh

Employee ⋈ Department (this is equivalent to ***Employee*** ⋈ **$_{Emp.Dept=Depart.Dept}$** ***Department***)

e-id	e-name	Dept	manager
11	Bhupi	Computer	Anisha
13	Anju	Finance	Manisha
43	Manju	Computer	Anisha
54	Nisha	Finance	Manisha

Note:- The natural join is also called equijoin.

Theta join operation:

The theta join operation is an extension to the natural join operation that allows us to specify the join condition. The theta condition consists one of the comparison operators {=, <, <=, >, >=, < >}. When join condition is = i.e. θ is =, the operation is called an equijoin.

Example:

Employee

e-id	e-name	salary
11	Bhupi	3000
13	Anju	4000
43	Manju	5000
54	Nisha	6000

Department

Dept-id	Manager
09	Anisha
22	Manisha
59	Umesh

Then ***Employee*** ⋈ ***$_{e\text{-}id > Dept\text{-}id}$ Department*** is

e-id	e-name	salary	Dept-id	Manager
11	Bhupi	3000	09	Anisha
13	Anju	4000	09	Anisha
43	Manju	5000	09	Anisha
43	Manju	5000	22	Manisha
54	Nisha	6000	09	Anisha
54	Nisha	6000	22	Manisha

2.3 Division operation (÷):

It is denoted by symbol ÷ and is suited to queries that include the phrase "for all". It takes two relations and builds another relation, consisting of values of an attribute of one relation that match all the values in the other relation.

Examples of Division A÷B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Example 2: let's take two relations Depositor and Branch as below:

Depositor

customer-name	account-number
Pukar	A-101
Shikha	A-102
Anisha	A-201
Gaurab	A-209
Bikky	A-233
Binek	A-409
Kamala	A-511

Branch

branch-name	branch-city	Assets
Newroad-branch	Kathmandu	7000
Pokhara-branch	Pokhara	3000
Kirtipur-branch	Kirtipur	9000
Dodhara-branch	Lalitpur	6000
Kalanki-branch	Kathmandu	7200
Balkhu-ranch	Kathmandu	2200
Banepa-branch	Banepa	4000

Account

Account-number	branch-name	Balance
A-101	Newroad-branch	50000
A-102	Kirtipur-branch	60000
A-201	Balkhu-branch	90000
A-206	Pokhara-branch	20000
A-301	Kalanki-branch	12000
A-401	Banepa-branch	22000
A-503	Dodhara-branch	41000

Suppose we want to find all the customers who have an account at **all** branches located in Kathmandu.
Strategy: think of it as three steps.
We can obtain the names of all branches located in Kathmandu by

$$r1= \prod_{bname}(\sigma_{bcity="Kathmandu"} (branch))$$

branch-name
Newroad-branch
Kalanki-branch
Balkhu-ranch

We can also find all *cname*, *bname* pairs for which the customer has an account by

$$r2=\prod_{cname, bname}(depositor \bowtie account)$$

customer-name	branch-name
Pukar	Newroad-branch
Shikha	Kirtipur-branch
Anisha	Balkhu-branch

Now we need to find all customers who have an account at **all** branches located in Kathmandu.
The divide operation provides exactly those customers:

$$\prod_{cname, bname}(depositor \bowtie account) \div \prod_{bname}(\sigma_{bcity="Kathmandu"} (branch))$$

customer-name
Pukar
Shikha
Anisha

2.4 The Assignment Operation:

The assignment operation (←) provides a convenient way to express complex queries. It helps human beings with writing out complex relational expressions in steps so that they can be more easily understood.
The assignment operation denoted by ← and works like assignment in a programming language.
Example:
Variable ← E, Where E is any relational algebra expression.

3. Extended Relational Algebra Operations

3.1 Outer join operation:

The *Outer join* operation is an extension of the join operation to deal with missing information.
Three types of outer joins:

3.1.1 Left outer join operation (⟕):

It includes all tuples in the left hand relation and includes only those matching tuples from the right hand relation.

Example:

Assume we have two relations: PEOPLE and MENU:

PEOPLE:

Name	Age	Food
Alice	21	Hamburger
Bill	24	Pizza
Carl	23	Beer
Dina	19	Shrimp

MENU:

Food	Day
Pizza	Monday
Hamburger	Tuesday
Chicken	Wednesday
Pasta	Thursday
Tacos	Friday

Then PEOPLE ⟕ MENU is

Name	Age	people.Food	menu.Food	Day
Alice	21	Hamburger	Hamburger	Tuesday
Bill	24	Pizza	Pizza	Monday
Carl	23	Beer	*NULL*	*NULL*
Dina	19	Shrimp	*NULL*	*NULL*

3.1.2 Right outer join (⟖):

It includes all tuples in the right hand relation and includes only those matching tuples from the left hand relation.

Example:

Assume we have two relations: PEOPLE and MENU as above:

Then PEOPLE ⟖ MENU is

Name	Age	people.Food	menu.Food	Day
Bill	24	Pizza	Pizza	Monday
Alice	21	Hamburger	Hamburger	Tuesday
NULL	*NULL*	*NULL*	Chicken	Wednesday
NULL	*NULL*	*NULL*	Pasta	Thursday
NULL	*NULL*	*NULL*	Tacos	Friday

3.1.3 Full outer join (⟗):

It includes all tuples in the left hand relation and from the right hand relation.

Example:

Assume we have two relations: PEOPLE and MENU as above:

Then PEOPLE ⟗ MENU is

Name	Age	people.Food	menu.Food	Day
Alice	21	Hamburger	Hamburger	Tuesday
Bill	24	Pizza	Pizza	Monday
Carl	23	Beer	*NULL*	*NULL*
Dina	19	Shrimp	*NULL*	*NULL*
NULL	*NULL*	*NULL*	Chicken	Wednesday
NULL	*NULL*	*NULL*	Pasta	Thursday
NULL	*NULL*	*NULL*	Tacos	Friday

3.2 Null values:

It is possible for tuples to have a null value, denoted by *null*, for some of their attributes

- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*.
- Aggregate functions simply ignore null values (as in SQL)
- iFor duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

3.3 Generalized projection:

It extends the projection operation by allowing arithmetic functions to be used in the projection list. The generalized projection operation has the form:

$$\prod_{F1,\ F2,\ldots,\ Fn} (E)$$

Where E is any relational-algebra expression, and each of *F*1, *F*2, …, *Fn* are arithmetic expressions involving constants and attributes in the schema of *E*.

Example: Given relation ***instructor****(ID, name, dept_name,* salary)

Where salary is annual salary, get the same information but with monthly salary we use following projection operation called generalized projection.

$$\prod_{ID,\ name,\ dept_name,\ salary/12} (\textbf{instructor})$$

3.4 Aggregate Functions

Aggregate functions are functions that take a collection of values and return a single value as a result. It is denoted by symbol($\mathcal{G}$) read it as "calligraphic G".

Some aggregate functions are: **sum, avg, count, max, min**.

Example: let's take a relation "Fulltime-works" with a number of tuples as below:

Fulltime-works

employee-name	branch-name	Salary
Ram	Patan-branch	30000
Shyam	Tokha-branch	20000
Rehman	Palpa-branch	40000
Ram	Patan-branch	25000

Problem: "Suppose we want to find the total salary of all the full time employees in branch wise"

$$\text{branch-name}\ \mathcal{G}_{\text{sum}(\text{salary})}(\text{Fulltime-works})$$

The result of aggregate function with grouping specified above will be:

branch-name	sum of salary
Patan-branch	55000
Tokha-branch	20000
Palpa-branch	40000

Problem: Find the minimum Salary:

$$\mathcal{G}_{\text{min}(\text{salary})}(\text{Fulltime-works})$$

Results:

MIN(salary)
20000

Problem: Count the number of employees in the Patan-branch:

$$\mathcal{G}_{\text{count}_{(\text{employee-name})}}(\sigma_{\text{branch}-name = \text{Patan-branch}})(\text{Fulltime-works})$$

Results:

COUNT(employee-name)
2

Database Manipulation:

Until now we only did the extraction of information from the database. In this section we will perform some modification on the database. We will namely use three types of operations for the modification of the database; they are ***insertion, deletion*** and ***modification***.
All these operations can be expressed using the assignment operator.

Insertion operation:

To insert data into a relation, we specify a tuple to be inserted.
Syntax: R←R U E
Where R is a relation and E is a relational algebra expression.
Example: suppose we have a relation employee

Employee (Name, Salary, Address)

Suppose we wish to insert an employee "Bhupi" of salary 50,000 and live in Kathmandu then we write,

Employee ←Employee U {"Bhupi", 50000, Kathmandu}

Deletion operation:

We can remove the selected tuples from the database. We cannot delete values of only particular attributes.

Syntax: R←R−E
Where R is a relation and E is a relational algebra expression.
Example: Delete all of Anju information from Employee relation

Employee

e-id	e-name	Salary
11	Bhupi	3000
13	Anju	4000
43	Manju	5000
54	Nisha	6000
33	Anju	3400

Employee ← Employee − σ $_{\text{e-name="Anju"}}$ (Employee)

Result:

Employee

e-id	e-name	Salary
11	Bhupi	3000
43	Manju	5000
54	Nisha	6000

Updating Operation:

In some situation we may wish to change a value in tuple without changing all values in the tuple. We can use the generalized-projection operator to do this task.

Syntax: $\Pi_{<A1,\ A2,.....An>}$ *(Relation)*

Where {A1, A2,........,An} are attributes.

Example: All employees working in department "Computer" has increased their salary by 15%.

Employee

e-id	e-name	department	salary
11	Bhupi	Computer	3000
13	Anju	Math	4000
43	Manju	Physics	5000
54	Nisha	Computer	6000
33	Anisha	Math	6400

Employee $\leftarrow$ ($\Pi_{\text{e-id, e-name, department, salary + salary*0.15}}$ ($\sigma_{\text{department="Computer"}}$ (Employee)) U $\Pi_{\text{e-id, e-name, department, salary}}$ ($\sigma_{\text{department} \neq \text{"Computer"}}$ (Employee)))

Result: ***Employee***

e-id	e-name	department	salary
11	Bhupi	Computer	3450
13	Anju	Math	4000
43	Manju	Physics	5000
54	Nisha	Computer	6900
33	Anisha	Math	6400

Relational Algebra Examples:

Example 1: Consider the relational database:

employee (<u>person-name</u>, street, city)
works (<u>person-name</u>, company-name, salary)
company (<u>company-name</u>, city)
manages (<u>person-name</u>, manager-name)
Give an expression in relational algebra for each of following requests:

1. Find the name of all employees who works for "NIBL Bank ".

$$\Pi_{\text{person-name}} (\sigma_{\text{company-name="NIBL Bank"}} (\text{works}))$$

2. Find the names and cities of residence of all employees who work for "NIBL Bank".

$$\Pi_{\text{person-name, city}} (\sigma_{\text{company-name="NIBL Bank"}} (\text{employee} \bowtie \text{works}))$$

3. Find the names, street address, and cities of residence of all employees who works for "Software Company" and earn more than 50000 per month.

$$\Pi_{\text{person-name, street, city}} (\sigma_{\text{company-name="Software company"} \wedge \text{salary} > 50000}(\text{employee} \bowtie \text{works}))$$

4. Find the name of all employees in the database who live in the same city as the company for which they work.

$$\Pi_{\text{person-name}} (\text{employee} \bowtie \text{works} \bowtie \text{company}))$$

5. Find the name of all employees in the database who do not work for "SBI Bank".

$$\Pi_{\text{person-name}} (\sigma_{\text{company-name} \neq \text{"SBI Bank"}}(\text{works}))$$

6. Find the names of all employees who earn more than every employee of "SBI bank".

$$\text{Temp} \leftarrow \mathcal{G}_{\max(\text{salary})} (\sigma_{\text{company-name="SBI Bank"}} (\text{works}))$$

$$\Pi_{\text{person-name}} (\sigma_{\text{salary} > \text{Temp}} (\text{works}))$$

7. Assume the company may be located in several cities. Find all companies located in every city in which "SBI Bank" is located.

$$\Pi_{\text{company-name, city}} (\text{company}) \div \Pi_{\text{city}} (\sigma_{\text{company-name} = \text{"SBI Bank"}}(\text{company}))$$

8. Give all employees of "SBI Bank" a 15% salary rise.

$$\text{Works} \leftarrow \Pi_{\text{person-name, company-name, salary +salary*0.15}} (\sigma_{\text{company-name="SBI Bank"}} (\text{works}))$$

9. Delete all tuples in the employee relation where employee's city is "Kathmandu".

$$\text{employee} \leftarrow \text{employee} - (\sigma_{\text{city="Kathmandu"}} (\text{employee}))$$

Example 2: Consider the relational database:

Unit 4

Structured Query Language (SQL)

- Basic structure
- Set operation
- Aggregate functions
- NULL values
- Nested sub queries
- Views
- Modification of database
- joined relations
- Data definition languages (DDL)
- Other SQL features:

  Dynamic and Embedded SQL

Introduction:

SQL is a computer language for organizing, managing, and retrieving data stored by a computer database. In fact, SQL works with one specific type of database, called ***a relational database***. The name "SQL" is the short form for ***Structured Query Language***.

SQL is used to control all of the functions that a DBMS provides for its users, including:

1. **Data definition:** SQL lets a user define the structure and organization of the stored data and relationships among the stored data items.
2. **Data retrieval:** SQL allows a user or an application program to retrieve stored data from the database and use it.
3. **Data manipulation:** SQL allows a user or an application program to update the database by adding new data, removing old data, and modifying previously stored data.
4. **Access control:** SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.
5. **Data sharing:** SQL is used to coordinate data sharing by concurrent users, ensuring that they do not interfere with one another.
6. **Data integrity:** SQL defines integrity constraints in the database, protecting it from corruption due to inconsistent updates or system failures.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database

SQL DML and DDL:

SQL can be divided into two parts: The Data Manipulation Language (DML) and the Data Definition Language (DDL).

The query and update commands form the DML part of SQL:

- **SELECT:** - extracts data from a database
- **UPDATE:** - updates data in a database
- **DELETE:**- deletes data from a database
- **INSERT INTO: -** inserts new data into a database

The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specify links between tables, and impose constraints between tables.

The most important DDL statements in SQL are:

- **CREATE DATABASE**- creates a new database
- **ALTER DATABASE**- modifies a database
- **CREATE TABLE**- creates a new table
- **ALTER TABLE**- modifies a table
- **DROP TABLE**- deletes a table
- **CREATE INDEX**- creates an index (search key)
- **DROP INDEX**- deletes an index

Basic structure:

The basic structure of an SQL expression consists of three clauses: **SELECT, FROM** and **WHERE.**

- The SELECT clause corresponds to the projection operation of relational algebra. It is used to list the attributes desired in the result of a query.
- The FROM clause corresponds to the Cartesian product operation of relational algebra. It is used to list the relations to be used in the evaluation of the expression.
- The WHERE clause corresponds to the selection predicate of relational algebra. It consists of a predicate in the attributes of the relations that appear in the FROM clause.

A typical SQL query has the form

SELECT $A_1, A_2, \ldots, A_n$
FROM $R_1, R_2, \ldots, R_n$
WHERE P

Where each A_i represents an attribute, each R_i is a relation and P is a predicate.

Its equivalent relational algebra expression is:

$$\prod_{A1, A2, \ldots, An} (\sigma_p (R_1 \times R_2 \times \ldots \times R_n))$$

The SQL SELECT Statement:

The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set.

SQL SELECT Syntax:

SELECT column_name(s) FROM table_name

and

SELECT * FROM table_name

> Note: SQL is not case sensitive. SELECT is the same as select.

An SQL SELECT Example:

Sailors (<u>*sid:* integer</u>, *sname:* string, *rating:* integer, *age:* real)
Boats (<u>*bid:* integer</u>, *bname:* string, *color:* string)
Reserves (*sid:* integer, *bid:* integer, *day:* date

Sailors

Sid	sname	Rating	age
1	Ajaya	12	33
2	Robin	11	43
3	Ganga	32	28
4	Manoj	9	31
7	Rahul	7	22
9	Sanjaya	9	42
11	Raju	4	19

Boats

Bid	Bname	color
11	Marine	Red
24	Clipper	Blue
33	Wooden	Black
41	Marine	Green

Reserves

Sid	bid	Day
1	24	2068-08-11
1	11	2068-08-11
1	41	2068-08-22
2	33	2068-11-08
2	11	2068-08-19
11	41	2068-09-23
9	24	2068-08-10
9	11	2069-08-15
9	33	2068-05-21

Now we want to select the content of the columns named "sname" and "age" from the table Sailors. We use the following SELECT statement:

SELECT sname, age
FROM Sailors;

The result-set will look like this:

Sname	Age
Ajaya	33
Robin	43
Ganga	28
Manoj	31
Rahul	22
Sanjaya	42
Raju	19

SELECT * Example

Now we want to select all the columns from the “Sailors " table. We use the following SELECT statement:

*SELECT * FROM Sailors*

Tip: The asterisk (*) is a quick way of selecting all columns!

The result-set will look like this:

Sid	sname	Rating	age
1	Ajaya	12	33
2	Robin	11	43
3	Ganga	32	28
4	Manoj	9	31
7	Rahul	7	22
9	Sanjaya	9	42
11	Raju	4	19

The SQL SELECT DISTINCT Statement:

In a table, some of the columns may contain duplicate values. This is not a problem; however, sometimes you will want to list only the different (distinct) values in a table. The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax:

SELECT DISTINCT column_name(s)
FROM table_name

SELECT DISTINCT Example:

Now we want to select only the distinct values from the column named "bname" from the table “Boats”. We use the following SELECT statement:

SELECT DISTINCT bname FROM Boats

The result-set will look like this:

Bname
Marine
Clipper
Wooden

The WHERE Clause:

The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax:

SELECT column_name(s)
FROM table_name
WHERE column_name operator value

WHERE Clause Example:

Now we want to select only those Sailors whose age is less than 30 from the table Sailors above. We use the following SELECT statement:

SELECT *
FROM Sailors
WHERE age < 30;

The result-set will look like this:

Sid	sname	Rating	age
3	Ganga	32	28
7	Rahul	7	22
11	Raju	4	19

Quotes around Text Fields:

SQL uses single quotes around text values (most database systems will also accept double quotes). Although, numeric values should not be enclosed in quotes.

For text values:

This is correct: SELECT *

FROM Sailors
WHERE sname='Ajaya';
This is wrong: SELECT *
FROM Sailors
WHERE sname=Ajaya;

For Numeric values:
This is correct: SELECT *
FROM Sailors
WHERE age=32
This is wrong: SELECT *
FROM Sailors
WHERE age='32'

Operators Allowed in the WHERE Clause:

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
< >	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns.
AND	And
OR	Or

Note: In some versions of SQL the <> operator may be written as !=

The AND & OR Operators:

The AND & OR operators are used to filter records based on more than one condition.

- The AND operator displays a record if both the first condition and the second condition is true.
- The OR operator displays a record if either the first condition or the second condition is true.

AND Operator Example:

Suppose we want to select only the Sailors with the name equal to "Ajaya" AND the age equal to 33: We use the following SELECT statement:

SELECT *
FROM Sailors
WHERE sname='Ajaya' AND age=33;

The result-set will look like this:

Sid	sname	rating	age
1	Ajaya	12	33

Example 2: *Find the sids of sailors who have reserved a red boat.*

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE B.bid = R.bid AND B.color = `red'
```

The result-set will look like this:

Sid
1
2
9

OR Operator Example:

Now we want to select only the Sailors with the first name equal to "Rahul" OR the rating equal to 9: We use the following SELECT statement:

```
SELECT *
FROM Sailors
WHERE sname='Rahul' OR rating=9;
```

The result-set will look like this:

sid	Sname	Rating	Age
4	Manoj	9	31
7	Rahul	7	22
9	Sanjaya	9	42

Combining AND & OR:

We can also combine AND and OR (use parenthesis to form complex expressions).

Now we want to select only the Sailors of rating equal to 9 AND the age equal to 31 OR to 42: We use the following SELECT statement:

```
SELECT *
FROM Sailors
WHERE rating=9 AND (age=31 OR age=42)
```

The result-set will look like this:

sid	Sname	Rating	Age
4	Manoj	9	31
9	Sanjaya	9	42

The ORDER BY Keyword:

The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sorts the records in ascending order by default. If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC | DESC
```

ORDER BY Example:

Now we want to select all the Sailors from the table above, however, we want to sort the Sailors by their name. We use the following SELECT statement:

```
SELECT *
 FROM Sailors
ORDER BY sname;
```

The result-set will look like this:

Sid	Sname	Rating	Age
1	Ajaya	12	33
3	Ganga	32	28
4	Manoj	9	31
7	Rahul	7	22
11	Raju	4	19
2	Robin	11	43
9	Sanjaya	9	42

ORDER BY DESC Example:

Now we want to select all the Sailors from the table above, however, we want to sort the Sailors descending by their name. We use the following SELECT statement:

```
SELECT *
FROM Sailors
ORDER BY sname DESC
```

The result-set will look like this:

Sid	Sname	Rating	Age
9	Sanjaya	9	42
2	Robin	11	43
11	Raju	4	19
7	Rahul	7	22
4	Manoj	9	31
3	Ganga	32	28
1	Ajaya	12	33

The SQL BETWEEN Operator:

The BETWEEN operator is used to select values within a range. The values can be numbers, text, or dates.

SQL BETWEEN Syntax:

SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;

BETWEEN Operator Example:

The following SQL statement selects all Sailors with age BETWEEN 20 and 40:

SELECT *
FROM Sailors
WHERE age BETWEEN 20 AND 40;

The result-set will look like this:

Sid	Sname	Rating	Age
7	Rahul	7	22
4	Manoj	9	31
3	Ganga	32	28
1	Ajaya	12	33

NOT BETWEEN Operator Example:

To display the Sailors outside the range of the previous example, use NOT BETWEEN:

SELECT *
FROM Sailors
WHERE age NOT BETWEEN 20 AND 40;

The result-set will look like this:

Sid	Sname	Rating	Age
9	Sanjaya	9	42
2	Robin	11	43
11	Raju	4	19

SQL IN Operator:

The IN operator allows us to specify multiple values in a WHERE clause.

SQL IN Syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1*, *value2...)*;

IN Operator Example:

The following SQL statement selects all Sailors with a rating of 9 or 11:

SELECT *
FROM Sailors
WHERE Rating IN (9, 11);

→ Its equivalent query by using OR operator is as below:

SELECT *
FROM Sailors
WHERE Rating=9 OR Rating=11;

The result-set will look like this:

Sid	sname	Rating	Age
2	Robin	11	43
4	Manoj	9	31
9	Sanjaya	9	42

NOT IN Operator Example:

The following SQL statement selects all Sailors with age not 11 or 9:

SELECT *
FROM Sailors
WHERE rating NOT IN (11, 9);

The result-set will look like this:

Sid	sname	Rating	age
1	Ajaya	12	33
3	Ganga	32	28
7	Rahul	7	22
11	Raju	4	19

String operations:

SQL specifies strings by enclosing them in single quotes, for example 'Pokhara'. The most commonly used operation on strings is pattern matching. It uses the operator ***LIKE.*** We describe the patterns by using two special characters:

- Percent (%): The % character matches any substring, even the empty string.
- Underscore (_): The underscore stands for exactly one character. It matches any character.

To illustrate pattern matching, we consider the following examples:

- 'A_Z': All string that starts with 'A', another character and end with 'Z'. For example, 'ABZ' and 'A2Z' both satisfy this condition but 'ABHZ' does not because between A and Z there are two characters are present instead of one.
- 'ABC%': All strings that start with 'ABC'.
- '%ABC': All strings that ends with 'ABC'.
- '%AN%': All strings that contains the pattern 'AN' anywhere. For example, 'ANGELS' ,'SAN', 'FRANCISCO' etc.
- '_ _ _': matches any strings of exactly three characters.
- '_ _ _%': matches any strings of at least three characters.

Example:

```
SELECT *
FROM Sailors
WHERE sname LIKE '%ya';
```

This SQL statement will match any Sailors first names that end with 'ya'.

The result-set will look like this:

Sid	sname	Rating	age
1	Ajaya	12	33
9	Sanjaya	9	42

Set Operations:

Some time it is useful to combine query results from two or more queries into a single result. SQL supports three set operators which are:

- SQL Union
- SQL Intersection and
- SQL Except (Minus)

These operators have the pattern:

<query1> <set operator> <query2>

SQL Union Operation:

In SQL the **UNION** clause combines the results of two SQL queries into a single table of all matching rows. The two queries must result in the same number of columns and compatible data types in order to unite. Any duplicate records are automatically removed unless UNION ALL is used.

Example: ***Find the names of sailors who have reserved a red or a green boat.***

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND (B.color = `red' OR B.color = `green')
```

This query is difficult to understand (and also quite inefficient to execute, as it turns out). A better solution for this query is to use UNION as follows:

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
UNION
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green'
```

The result-set will look like this:

Sname
Sanjaya
Robin
Raju
Ajaya

UNION ALL gives different results, because it will not eliminate duplicates. Executing this statement:

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
UNION ALL
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green'
```

The result-set will look like this:

Sname
Sanjaya
Robin
Raju
Ajaya
Ajaya

INTERSECT Operation:

The SQL INTERSECT operator takes the results of two queries and returns only rows that appear in both result sets. The INTERSECT operator removes duplicate rows from the final result set. The INTERSECT ALL operator does not remove duplicate rows from the final result set.

Example: ***Find the names of sailors who have reserved a red and a green boat.***

```
SELECT S.sname
FROM Sailors S, Reserves R1, Boats B1, Reserves R2, Boats B2
WHERE S.sid = R1.sid AND R1.bid = B1.bid AND S.sid = R2.sid AND R2.bid = B2.bid
AND B.color = `red' AND B.color = `green';
```

This query is difficult to understand (and also quite inefficient to execute, as it turns out). A better solution for this query is to use INTERSECT as follows:

The result-set will look like this:

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
INTERSECT
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green';
```

The result-set will look like this:

Sname
Ajaya

Except Operation:

The SQL EXCEPT operator takes the distinct rows of one query and returns the rows that do not appear in a second result set.

Example: ***Find the names of sailors who have reserved a red boats but not a green boat.***

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
EXCEPT
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green'
```

The EXCEPT operation automatically eliminate duplicates. If we want to retain all duplicates, we must write EXCEPT ALL in place of EXCEPT.

SQL Aggregate Functions:

Aggregate functions are functions that take a collection of values as input and return a single value.

Useful aggregate functions are:

- SUM() - Returns the sum
- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- FIRST() - Returns the first value
- LAST() - Returns the last value

Example 1: find sum of rating of all sailors.

```
SELECT SUM (rating)
FROM Sailors;
```

The result-set will look like this:

Sum(rating)
85

Example 2: find average age of all sailors.

```
SELECT AVG (age)
FROM Sailors;
```

The result-set will look like this:

Avg(age)
29.7143

Example 3: find average age of all sailors with a rating of 9.

```
SELECT AVG (age)
FROM Sailors
WHERE rating=9;
```

The result-set will look like this:

Avg(age)
31.5000

Example 4: find name and age of oldest sailor.

```
SELECT sname, age
FROM Sailors
WHERE age = (SELECT MAX(age)
             FROM Sailors);
```

The result-set will look like this:

Sname	max(age)
Robin	43

Example 5: count number of sailors.

```
SELECT COUNT(*)
FROM Sailors;
```

The result-set will look like this:

count(*)
7

Example 6: Find the names of sailors who are older than the oldest sailor with a rating of 9.

```
SELECT S.sname
FROM Sailors S
WHERE S.age > (SELECT MAX (S2.age)
FROM Sailors S2
WHERE S2.rating =9);
```

The result-set will look like this:

Sname
Robin

Example 7: find the maximum and minimum aged sailors name.

```
SELECT sname
FROM Sailors
WHERE age=(SELECT  max(age) FROM Sailors)
```

```
UNION
SELECT S1.sname
FROM Sailors S1
WHERE S1.age=(SELECT  min(age) FROM Sailors);
```

The result-set will look like this:

Sname
Robin
Raju

GROUP BY Clause:

The SQL GROUP BY clause is use to divide the rows in a table into groups. The GROUP BY statement is used along with the SQL aggregate functions. In GROUP BY clause, the tuples with same values are placed in one group.

Example: ***Find the age of the youngest sailor for each rating level.***

```
SELECT rating, MIN(age)
FROM Sailors
GROUP BY rating;
```

The result-set will look like this:

Rating	Min(age)
4	19
7	22
9	31
11	43
12	33
32	28

This table display the minimum age of each group according to their rating.

SQL HAVING Clause:

The SQL HAVING clause allows us to specify conditions on the rows for each group. It is used instead of the WHERE clause when Aggregate Functions are used. HAVING clause should follow the GROUP BY clause if we are using it.

Example: let’s take an instance S3 of Sailors,

S3

Sid	sname	Rating	age
1	Ajaya	12	33
2	Robin	11	43
3	Ganga	32	28
4	Manoj	9	31
7	Rahul	7	22
9	Sanjaya	9	42
11	Raju	4	19
22	Robin	11	54
32	Anish	7	21

Example: Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

The result-set will look like this:

Rating	Minage
7	21
9	31
11	43

Example 2: Find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two sailors.

```
SELECT S.rating, AVG ( S.age )
FROM Sailors S
WHERE S. age >= 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
            FROM Sailors S2
            WHERE S.rating = S2.rating );
```

NULL Values:

SQL allows the use of NULL values to indicate absence of information about the value of an attribute. It has a special meaning in the database- the value of the column is not currently known but its value may be known at a later time.

A special comparison operator IS NULL is used to test a column value for NULL. It has following general format:

```
Value1 IS [NOT] NULL;
```

This comparison operator return *true* if value contains NULL, otherwise return *false*. The optional NOT reverses the result.

Following syntax is illegal in SQL:

```
WHERE attribute=NULL;
```

Example: let's take an instance S4 of Sailors,

S4

sid	sname	Rating	Age
1	Ajaya	12	33
2	Robin	11	43
3	Ganga	32	28
4	Manoj	9	31
7	Rahul	7	22
9	Sanjaya	NULL	NULL
11	Raju	4	19
22	Robin	NULL	NULL
32	Anish	NULL	NULL

Find all sailors that appear in S4 relation with NULL values for rating and age:

```
SELECT sname
FROM S4
WHERE rating IS NULL AND age IS NULL;
```

The result-set will look like this:

sname
Sanjaya
Robin
Anish

Nested Sub-queries:

A nested query is a query that has another query embedded within it; the embedded query is called a sub-query. The result of sub query is used by the main query (outer query). We can place the sub-query in a number of SQL clauses including:

- The WHERE clause
- The HAVING clause
- The FROM clause

A common use of sub-queries is to perform tasks for set membership and make set comparison.

Set Membership:

The IN connective is used to test a set membership, where set is a collection of values produced by SELECT clause in sub-query. The NOT IN connective is used to test for absence of set membership.

Example1: Find the names of sailors who have reserved boat 41.

```
SELECT sname
FROM  Sailors
WHERE sid IN ( SELECT sid
               FROM Reserves
               WHERE bid=41);
```

The result-set will look like this:

Sname
Ajaya
Raju

Example 2: Find the names of sailors who have reserved a red boat.

```
SELECT sname
FROM sailors
WHERE sid IN (SELECT sid
             FROM Reservs
             WHERE bid IN ( SELECT bid
                            FROM Boats
                            WHERE color='Red'));
```

The result-set will look like this:

sname
Ajaya
Robin
Sanjaya

Example 3: Find the names of sailors who have not reserved a red boat.

```
SELECT sname
FROM sailors
WHERE sid NOT IN (SELECT sid
                  FROM Reservs
                  WHERE bid IN ( SELECT bid
                                 FROM Boats
                                 WHERE color='Red'));
```

The result-set will look like this:

Sname
Ganga
Manoj
Rahul
Raju

Set Comparison:

The comparison operators are used to compare sets in nested sub-query. SQL allows following set comparisons:

< SOME, <= SOME, > SOME, >= SOME, = SOME, < > SOME

<ALL, <=ALL, >ALL, >=ALL, = ALL, < >ALL

The keyword ANY is synonymous to SOME in SQL.

Example 1: let's take an instance S4 of Sailors as:

S4

Sid	Sname	Rating	Age
1	Ajaya	12	33
2	Robin	11	43
3	Ganga	32	28
4	Manoj	9	31
7	Rahul	7	22
9	Sanjaya	9	42
11	Raju	4	19
8	Rahul	6	76

Find the id and names of sailors whose rating is better than some sailor called "Rahul".

```
SELECT sid, sname
FROM S4
WHERE rating >ANY (SELECT rating
                   FROM S4
                   WHERE sname='Rahul');
```

The result-set will look like this:

Sid	Sname
1	Ajaya
2	Robin
3	Ganga
4	Manoj
7	Rahul
9	Sanjaya

Example 2: Find the id and names of sailors whose rating is better than every sailor called "Rahul".

```
SELECT sid, sname
FROM S4
WHERE rating >ALL (SELECT rating
                   FROM S4
                   WHERE sname='Rahul');
```

The result-set will look like this:

Sid	Sname
1	Ajaya
2	Robin
3	Ganga
4	Manoj
9	Sanjaya

Example 3: Find the id and name of sailor with height rating.

```
SELECT sid, sname
FROM S4
WHERE rating >=ALL (SELECT rating
                    FROM S4);
```

The result-set will look like this:

Sid	Sname
3	Ganga

Note: IN and NOT IN are equivalent to =ANY and < > respectively.

Views:

A database view is a logical table. It does not physically store data like tables but represent data stored in underlying tables in different formats. A view does not require desk space and we can use view in most places where a table can be used.

Since the views are derived from other tables thus when the data in its source tables are updated, the view reflects the updates as well. They also can be used by DBA to enforce database security.

Advantages of Views:

- Database security: view allows users to access only those sections of database that directly concerns them.
- View provides data independence.
- Easier querying
- Shielding from change
- Views provide group of users to access the data according to their criteria.
- Vies allow the same data to be seen by different users in different ways at the same time.

Syntax for creating view is: Optional

CREATE VIEW <view name> <columns> AS <query expression>

Where, <query expression> is any legal query expression.

Example: Following view contains the id, name, rating+5 and age of those Sailors whose age is greater than 30:

CREATE VIEW Sailor_view AS
SELECT sid, sname, rating+5, age
FROM Sailors
WHERE age>30;

Now by executing this query we get following view (logical table);

Sailor_view

Sid	sname	Rating+5	age
1	Ajaya	17	33
2	Robin	16	43
9	Sanjaya	14	42
8	Rahul	11	76

Now any valid database operations can be performed in this view like in that of general table.

Modification of the database:

Until now we only study about how information can be extract from the database. Now, we show how to add, remove, or change information with SQL.

To modify database, there are mainly three operations are used:

- ***Insertion***
- ***Deletion and***
- ***Updates***

1.Insertion:

To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted.

Example 1: suppose we need to insert a new record of Sailors of id is11, name is "Rahul", rating is 9 and of age is 29 then we write following SQL query,

INSERT INTO Sailors
VALUES (11,'Rahul', 9, 29);
OR
INSERT INTO Sailors (sid, sname, rating, age)
VALUES (11,'Rahul', 9, 29);

More generally, we might want to insert tuples on the basis of the result of query.

Example 2: suppose we have already some tuples on the relation 'Sailores'. Suppose we need to insert those tuples of sailors into their own relation whose rating is less than 7, this can be write as,

INSERT INTO Sailors
*SELECT **
FROM Sailors

WHERE rating<7;

2. Deletion:

It is used to remove whole records or rows from the table.

Syntax:

DELETE FROM table_name
WHERE <predicate>

Example 1: suppose we need to remove all tuples of Sailors whose age is 32,

DELETE FROM Sailors
WHERE age=32;

Example 2: Remove all tuples of Sailors whose age is less than 30 and rating greater than 7,

DELETE FROM Sailors
WHERE age<32 AND rating>7;

3. Updates:

If we need to change a particular value in a tuple without changing all values in the tuple, then for this purpose we use update operation.

Syntax:

UPDATE table_name
SET <column i> = <expression i>;

Example: suppose we need to increase the rating of those sailors whose age is greater than 40 by 20%, this can be write as,

UPDATE sailors
*SET rating=rating + rating*0.2*
WHERE age>40;

Joined relations:

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

The types the different SQL JOINs are:

- **INNER JOIN**
- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

INNER JOIN:

It is most common type of join. An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

SQL INNER JOIN Syntax:

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name=table2.column_name*;

Note: INNER JOIN is the same as JOIN.

Example 1: Find the sailor id, boat id, boat name, boat color of those sailors who have reserved a red boat.

SELECT sailors.sid, boats.bid, boats.bid, boats.bname, boats.color
FROM sailors INNER JOIN reserver INNER JOIN boats
WHERE sailors.sid=reserver.sid AND reserver.bid=boats.bid;

The result-set will look like this:

sid	sname	bid	bname	color
1	Ajaya	24	Clipper	Blue
1	Ajaya	11	Marine	Red
1	Ajaya	41	Marine	Green
2	Robin	33	Wooden	Black
2	Robin	11	Marine	Red
11	Raju	41	Marine	Green
9	Sanjaya	24	Clipper	Blue
9	Sanjaya	11	Marine	Red
9	Sanjaya	33	Wooden	Black

Example 2: Find the name and age of those sailors who have reserved a Marine boat.

SELECT sailors.sname, sailors.age
FROM sailors INNER JOIN reserver INNER JOIN boats
WHERE sailors.sid=reserver.sid AND reserver.bid=boats.bid;

The result-set will look like this:

sname	age
Ajaya	33
Ajaya	33
Ajaya	33
Robin	43
Robin	43
Raju	19
Sanjaya	42
Sanjaya	42
Sanjaya	42

LEFT OUTER JOIN:

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

SQL LEFT JOIN Syntax:

SELECT *column_name(s)*
FROM *table1*
LEFT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;

RIGHT OUTER JOIN:

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

SQL RIGHT JOIN Syntax:

SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;

<u>FULL OUTER JOIN:</u>

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2). The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

Data definition languages:

The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specify links between tables, and impose constraints between tables.

The most important DDL statements in SQL are:

- **CREATE DATABASE**- creates a new database
- **ALTER DATABASE**- modifies a database
- **CREATE TABLE**- creates a new table
- **ALTER TABLE**- modifies a table
- **DROP TABLE**- deletes a table
- **CREATE INDEX**- creates an index (search key)
- **DROP INDEX**- deletes an index

Domain type (data type) in SQL:

When we create a table each column of the table must be specified by their domain or data type. Due to e it helps us what type of data will be stored in the field.

The SQL standard supports a variety of build-in domain types which are:

- **Char (n):** A fixed length character data (string). Also we can use full form ***character***.
- **Varchar(n):** A variable character string.
- **Int:** used to represent whole number. Also we can use it's full form ***integer.***
- **Numeric(p, d):** A fixed point number with user-specified precision. The number consists of p digits (plus a sign), and d represent the number of digits to right of decimal point.
- **Real, double precision:** floating point numbers with machine dependent precisions.
- **Float(n):** floating point number with precision of at least n digits.
- **Date:** A calendar date containing a four digit year, month and day. Eg '2006-04-22'
- **Time:** The time of day, in hours, minutes, and seconds.eg '09:34:23'
- **Timestamp:** combination of date and time.eg '2008-05-21 11:23:08'

CREATE DATABASE:

The ***CREATE DATABASE*** statement is used to create a database.

Syntax:

CREATE DATABASE *dbname*;

Example:

CREATE DATABASE *my_db*;

After creating database 'my_db' we need to connect it as;
CONNECT my_db;

ALTER DATABASE:
Allow us to modify existing database name.
Syntax:

CREATE TABLE:
Allow us to create a new table within given database.
Syntax:

```
CREATE TABLE <table_name>
(
    <column1>    <data type> [not null] [unique][<integrity constraint>],
    <column2>    <data type> [not null] [unique][<integrity constraint>],
    ……………….
    ………………
    <column n>   <data type> [not null] [unique][<integrity constraint>]
)
```

Note: [ ]: optional

Example:

```
CREATE TABLE Sailors
(
    Sid     INTEGER NOT NULL,
    Sname   VARCHAR(12),
    Rating  INTEGER,
    Age     INTEGER,
    PRIMARY KEY (sid)
)
```

ALTER TABLE:
Allow us to modify a given table.
The structure of given table can be changed either of the following:

- By adding new column in existing table
- By deleting some columns from an existing table and
- By modifying some columns of given table

A new column can be added to the table as follows:
Syntax:

ALTER TABLE <table name>
ADD (<column_name> <datatype>);

Example: suppose we want to add a new column 'addresses' to an existing table Sailors,
ALTER TABLE Sailors

ADD (addresses varchar(15));

An existing column can be removed from the table as,

ALTER TABLE <table_name>
DROP (column_name);

Example: suppose we want to remove an existing column 'addresses' from the table sailors as,

ALTER TABLE Sailors
DROP (addresses);

An existing column can be modified as,

ALTER TABLE <table_name>
MODIFY (<column name> <data type>);

Example: modify sailors relation by changing the range of the name of sailors by 20,

ALTER TABLE sailors
MODIFY (sname varchar(20));

DROP TABLE

It allows us to remove an existing table from the database.
Syntax:

DROP TABLE <table name>

Example: if we want to remove a table 'Sailors' from the database my_db as,

DROP TABLE Sailors;

Embedded SQL:

The programming language in which SQL queries are embedded is called host language. And the SQL structures permitted in the host language is called embedded SQL. They are compiled by the embedded SQL processor.

Writing queries in SQL is usually much easier than coding same query in a programming language. However, a programmer must access to a database from a general purpose programming language for following two reasons:

- Not all queries can be expressed in SQL
- Non-declarative actions such as printing a report, interacting with user, or sending the result of query to a graphical user interface etc. cannot be done from the SQL.

Dynamic SQL:

The *dynamic SQL* component of SQL allows programs to construct and submit SQL queries at run time. In contrast, embedded SQL statements must be completely present at compile time; they are compiled by the embedded SQL preprocessor.

Exercise 1: Consider the following relations:

Student (*snum:* integer, *sname:* string, *major:* string, *level:* string, *age:* integer)
Class (c*name:* string, *meets-at:* time, *room:* string, *fid:* integer)
Enrolled (*snum:* integer, *cname:* string)
Faculty (fid: integer, *fname:* string, *deptid:* integer)

1. Find the names of all Juniors (Level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or is enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than _ve.
7. Print the Level and the average age of students for that Level, for each Level.
8. Print the Level and the average age of students for that Level, for all Levels except JR.
9. Find the names of students who are enrolled in the maximum number of classes.
10. Find the names of students who are not enrolled in any class.

Exercise 2 Consider the following schema:
Suppliers (*sid:* integer, *sname:* string, *address:* string)
Parts (*pid:* integer, *pname:* string, *color:* string)
Catalog (*sid:* integer, *pid:* integer, *cost:* real)
Write the following queries in SQL:

1. Find the *pname*s of parts for which there is some supplier.
2. Find the *sname*s of suppliers who supply every part.
3. Find the *sname*s of suppliers who supply every red part.
4. Find the *pname*s of parts supplied by Acme Widget Suppliers and by no one else.
5. Find the *sid*s of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).
7. Find the *sid*s of suppliers who supply only red parts.
8. Find the *sid*s of suppliers who supply a red part and a green part.
9. Find the *sid*s of suppliers who supply a red part or a green part.

Exercise 3 Consider the following schema of the relational database
Books (Bid,Btitle, Bauthor, Bpublisher, Bprice)
Members (Member_id, Name, Designation, Age)
Reserve (Member_id, Bid, Date)

1. Create the tables using Books, Members and Reserve by specifying the Primary key, Not NULL , Foreign key Constraints DDL Statement in MySQL database

2. Write SQL DML statement to insert any five tuples (Five records) in each relation(table)
3. Find the Books of Database System title and price above 500.
4. List the books published by TaTa McGraw Hill publication
5. Find the Name of the member who made reserve book in 12-10-2011

Exercise 4 Consider the following schema of the relational database

Department (dept_no,d_name, city)
Employee (emp_Id, e_name, salary)
Works (dept_no, emp_Id)

1. Create the above tables by specifying the Primary key, Not NULL , Foreign key Constraints DDL Statement in MySQL database
2. Write DML Statement to Insert any five records in each tables
3. Display the name of the employees.
4. Find the name of the employees whose salary is greater than 10000.
5. Find the department (d_name) of the employee '**Binek'**.

Exercise 5: Consider the following insurance database, where primary keys are underlined:

Teacher (Tid, Tname, Address, Age)
Student (Sid,Sname, Age, sex)
Takes (sid, course-id)
Course (course-id,course_name, text_book)
Teaches(Tid, Cousrse-id)
Taught-by{Sid, Tid}

Construct the following RA expressions for this relational database

a. Fine name, age and sex of all students who takes course "DBMS"
b. Find total number of students who are taught by the teacher "T01"
c. List all course names text books taught by teaher "T16"
d. Find average age of teachers for each course.
e. Insert the record of new teacher "T06" named "Bhupi" of age 27 into database who lives in "Balkhu" and takes course "DBMS"

Exercise 6: Consider the following employee database, where primary keys are underlined.

employee (employee-name, street, city, salary)
works (employee-name, company-name,)
company (company-name, city)
manages (employee-name, manager-name)

Give an expression in SQL for each of the following queries.

a. Find the names of all employees who work for Second Bank Corporation.
b. Find the names, street and cities of residence of all employees whose salary is more than average salary.
c. Find the names, street addresses, and cities of residence of all employees to whom company is not assigned yet.
d. Find the names of all employees who work under the manager "Devi Prasad".
e. Increase the salary of employees by 10% if their salary is less than 10,000 and by 8% otherwise.

Unit 5

Introduction of Database

- Domain constraints
- Referential constraints
- Triggers
- Assertion

Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database.

E-R model ensure two types of integrity constraints:

– **Key declaration**: Primary and candidate

– **Form of relationship**: many to many, one to many, one to one

<u>Types of constraints:</u>

<u>*Domain constraint:*</u>

Domain is a pool of values of the same type from which one or more attributes in one or more tables take their values.

D1 D2 Dn

Student

A1	A2	…………………	An

In the above student table, the attribute A1 draws value from domain D1, A2 from D2 and so on. Domain integrity means the definition of a valid set of values for an attribute. You define

- ✓ data type
- ✓ Length or size
- ✓ Is null value allowed
- ✓ Is the value unique or not for an attribute.

Domain constraints are the most elementary form of integrity constraint. They are tested essentially by the system whenever a new data item is entered into the database.

It is possible for several attributes to have the same domain.

- ✓ For example, the attributes *customer-name* and *employee-name* might have the same domain: ***the set of all person names***.

- ✓ However, the domains of *balance* and *branch-name* certainly ought to be distinct.
- ✓ It is perhaps less clear whether *customer-name* and *branch-name* should have the same domain.

At the implementation level, both customer names and branch names are character strings. However, we would normally not consider the query "Find all customers who have the same name as a branch" to be a meaningful query. Thus, if we view the database at the conceptual, rather than the physical, level, *customer-name* and *branch-name* should have distinct domains.

The **CREATE DOMAIN** clause can be used to define new domains. For example, to ensure that *rating* must be an integer in the range 1 to 10, we could use:

CREATE DOMAIN RATINGVAL INTEGER DEFAULT 0
CHECK (VALUE >= 1 AND VALUE <= 10)

Entity Integrity

The entity integrity constraint ensures that the primary key of a relation must be unique and not null.

Example: ***Employee***

cid	Cname	caddress	cphone
1	Abin	Kathmandu	9849248488
2	Anish	Lalitpur	9849245544
?	Binek	Kirtipur	9813334849

Entity integrity violation

Referential Integrity

Referential integrity ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation to establish the relationship between tables.

For referential integrity to hold in a relational database, any field in a table that is declared a foreign key can contain either a null value, or only values from a parent table's primary key. For instance, deleting a record that contains a value referred to by a foreign key in another table would break referential integrity.

In relational model we often store data in different tables and put them together to get complete example. For example, in PAYMENTS table we have only ROLLNO of the student. To get remaining information about the student we have to use STUDENTS table.

STUDENTS

RollNO	Name	Address
1	Anisha	Ktm
2	Bibek	Pokhara
3	Nikey	Lalitpur
4	Rashmi	Bhaktapur

PAYMENTS

RollNO	Date	Amount
1	12-03-2010	10000
3	06-08-2011	5000
2	02-07-2012	9000
4	14-03-2013	15000

Foreign Key

Thus for referential integrity a foreign key can have only two possible values- either the relevant primary key or a null value. No other values are allowed.

Referential Integrity in SQL
Primary and candidate keys and foreign keys can be specified as parts of the SQL **create table** statement:
Example:

CREATE TABLE ***Sailors***
(
sid ***integer not null***,
sname **varcha**r(20)**,**
rating **integer**,
age ***integer***,
primary key (sid*)*
)

CREATE TABLE ***Boats***
(
bid **integer not null,**
bname **varchar**(20),
color **varchar**(10),
primary key *(bid)*
)

CREATE TABLE ***Reserve***
(
sid **integer,**
bid **integer,**
rdate **date**,
foreign key (sid*)* **references** *Sailors,*
foreign key (bid*)* **references** *Boats*,
)

CHECK Constraints:

CHECK constraints allow users to prohibit an operation on a table that would violate the constraint. It is a local constraint.
Example: To ensure that *rating* must be an integer in the range 1 to 10, we could use:

```
CREATE TABLE Sailors ( sid      INTEGER,
                       sname    CHAR(10),
                       rating   INTEGER,
                       age      REAL,
                       PRIMARY KEY (sid),
                       CHECK ( rating >= 1 AND rating <= 10 ))
```

In sailors table if we are trying to insert a new record as
INSERT INTO Sailors
VALUES (5, "Bhupi", 15, 27.4);
We get insertion is rejected message since value of rating attribute violated the check condition.

Assertions:

Table constraints are associated with a single table, although the conditional expression in the CHECK clause can refer to other tables. Table constraints are required to hold only if the

associated table is nonempty. Thus, when a constraint involves two or more tables, the table constraint mechanism is sometimes cumbersome and not quite what is desired. To cover such situations, SQL supports the creation of **assertions**, which are constraints, not associated with any one table.

Assertion in the SQL takes the form

```
CREATE ASSERTION <assertion_name> CHECK<predicate>
```

Example, suppose that we wish to enforce the constraint that the number of boats plus the number of sailors should be less than 100.

```
CREATE ASSERTION SailorCheck
CHECK ((SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM Boats B) < 100);
```

Trigger:

A **trigger** is a procedure (statement) that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an **active database**. To design a trigger mechanism, we must meet following three requirements:

1. **Event**: A change to the database that **activates** the trigger.
2. **Condition**: A query or test that is run when the trigger is activated.
3. **Action**: A procedure that is executed when the trigger is activated and its condition is true.

Need for Triggers:

Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met. As an illustration, suppose that, instead of allowing negative account balances, the bank deals with overdrafts by setting the account balance to zero, and creating a loan in the amount of the overdraft. The bank gives this loan a loan number identical to the account number of the overdrawn account. For this example, the condition for executing the trigger is an update to the *account* relation that results in a negative *balance* value. Suppose that Jones' withdrawal of some money from an account made the account balance negative. Let *t* denote the account tuple with a negative *balance* value. The actions to be taken are: